UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

TRABAJO FIN DE MÁSTER

Diseño e implementación de una herramienta de interacción para productos de sistemas de comunicación sobre una FPGA

Máster Universitario en Ingeniería de Telecomunicación

Autor: Duque Casero, Domingo
Tutor: Aparicio Castrillo, Samuel
Ponente: Boemo Scalvinoni, Eduardo Iván

FECHA: Junio, 2021

Diseño e implementación de una herramienta de interacción para productos de sistemas de comunicación sobre una FPGA

AUTOR: Domingo Duque Casero
TUTOR: Samuel Aparicio Castrillo
PONENTE: Eduardo Iván Boemo Scalvinoni

SENER
Laboratorio de Sistemas Digitales
Dpto. Tecnología Electrónica y de Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio de 2021

Resumen

Este trabajo Fin de Máster tiene como objetivo la realización de un diseño que sirva como medio de estimulación para cierto producto de la empresa SENER. Cuenta con dos etapas: Transmisión y Recepción.

El marco tecnológico se limita a la empresa Xilinx. En particular, al programa Vivado 2019.1. junto a la placa de evaluación RFSoC ZCU111, la cual es necesaria para trabajar con aspectos de radiofrecuencia gracias a su cadena de RF. A su vez, permite la interacción con el mundo real (analógico) a través de sus DACs y ADCs.

Previo a la realización del diseño, se han establecido ciertos requerimientos que se deben cumplir:

- "Para la etapa de transmisión, se utilizan modelos de software para la generación de escenarios con una frecuencia de muestreo de 9.984 MHz".
- "El diseño utiliza 8 DACs para la transmisión de datos y 8 ADCs para recepción, todos los dispositivos funcionan con señales complejas. La resolución de las muestras es de 32 bits".
- "El diseño debe alcanzar al menos 3 GHz de ancho de banda tanto en transmisión como en recepción. Ambas etapas con la misma tasa de muestreo".
- "El diseño debe realizar ambas acciones (transmitir y recibir) "en coherencia", es decir, al unísono entre señales".
- "El control del diseño se debe realizar a partir de un host externo a elección del diseñador".

Una vez acabado el diseño, se han realizado pruebas para comprobar que el funcionamiento es correcto en todos los canales. Para ello, se ha necesitado de material de laboratorio como medio de visualización para monitorizar el comportamiento de la señal. Y como control del diseño en el lado del host, se ha trabajado con PYNQ que facilita su implementación a través de un simple *script*.

Como conclusión, se ha podido comprobar que esta placa solo está destinada en un principio para tareas de evaluación. Trae consigo acoplada una placa (XM500 RFMC Balun) para realizar la conversión de digital a analógico (DAC) y la conversión de analógico a digital (ADC), pero no contienen las mismas características. Sin embargo, gracias a una placa adaptada por la empresa SENER, donde la longitud de pista para todos los canales es "idéntica", se ha podido obtener resultados óptimos para la etapa de transmisión, alcanzando un desfase "casi nulo" a lo largo de todo el ancho de banda. Por otro lado, en la etapa de recepción, esto no ha sido posible, no existe una placa acondicionada actualmente. De modo que no se han obtenido resultados concluyentes. La longitud de pista entre los diferentes canales no es la misma, imposibilitando cumplir el requisito de "coherencia entre canales*"* (*multisincronización entre tiles*). Aun así, se ha realizado un análisis de desfase entre canales del mismo *tile*, para poder resolver el problema a través de una posible calibración. Sin embargo, esta compensación introducida no alcanza el nivel de precisión que se pretende.

Palabras clave: FPGA, convertidor de digital a analógico (DAC), conversor de analógico a digital (ADC), radiofrecuencia (RF), Vivado, *multisincronización entre tiles*, tasa de transferencia efectiva, tasa de muestreo.

Abstract

The aim of this Master's Thesis is to create a design that works as a way of interaction for a certain product of the SENER company. It has two stages: transmission and reception.

The technological framework is limited to the Xilinx company, especially, to the Vivado 2019.1 with RFSoC ZCU111 evaluation board, which is necessary to work with radiofrequency aspects due to its RF chain. At the same time, it allows the interaction with analogical signals through its DACs and ADCs.

Before starting the design, certain requirements that have been lay down must be fulfill:

- “For the transmission stage, software models are used to generate scenarios with a sampling frequency of 9.984 MHz".
- “The design uses 8 DACs for data transmission and 8 ADCs as reception, all devices work with complex signals. The resolution of the samples is 32 bits”.
- “The design must achieve at least 3 GHz of bandwidth both in transmission and reception. Both with the same sampling rate”.
- "The design must carry out both actions (transmit and receive) “with coherence”, that is, at the same time between signals".
- "Design control must be done from an external host on the designer's choice".

Once the design is concluded, tests have been made to verify a correct operation in all the channels. Also, laboratory equipment was necessary as a display to monitor the signal performance. As control design on the host side, it was used PYNQ, because it makes easier its implementation through a simple script.

To sum up, it has been found that this board it is only thought for evaluation tasks. The board brings with it another one which converts from digital to analogic (DAC) and from analogic to digital (ADC), but they do not share the same characteristics. Nevertheless, thanks to an adapted board by SENER company, where the track length for all channels is the same, it has been possible to obtain ideal results in transmission stage getting an almost zero mismatch during bandwidth. On the other hand, on reception stage, it has not been possible because it does not exist a proper board. That is the reason why conclusive results have not been obtained. The track length between the different channels it is not the same, making fulfil, the “coherence between channels” (*multi-tile synchronization*), requirement impossible. Even so, it has been carried out a mismatch analysis between channels from the same “tile” to be able to solve a problem through a possible calibration. But this introduced correction does not reach the level of accuracy planned.

Keywords: FPGA, digital to analog converter (DAC), analog to digital converter (ADC), radiofrequency (RF), Vivado, multi-tile synchronization, throughput, sampling rate.

Agradecimientos

En primer lugar, para esta segunda etapa en el máster, esta dedicatoria va especialmente a mis padres, José y Margarita, por su apoyo incondicional.

Agradecer principalmente a Julian Spahr (técnico de CRISA y extrabajador de SENER), fue realmente él quien me enseñó todo y lo he querido transmitir en este documento.

Gracias a Eduardo Boemo. A mis compañeros del Máster, Jorge y Darío. Y como no agradecérselo a ella, Irene Gasca, por estar ahí en todo momento.

Por último, a la empresa SENER. Gracias por contar conmigo en sus filas. Y en especial, a Samuel Aparicio (mi tutor), y a dos increíbles ingenieros, Francisco Javier Toral y Andrea Di Giovanni.

Gracias a todos.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE ECUACIONES

1 Introducción

1.1 Motivación

En los últimos años, la empresa SENER está invirtiendo una gran parte de sus recursos en el apartado de productos I+D. En concreto, en relación a este trabajo, al campo de la Inteligencia de Comunicaciones. De aquí el nombre del proyecto COMINT por sus siglas en inglés (*Communication Intelligence*).

El proyecto SENER COMINT trabaja para desarrollar soluciones orientadas en el apartado de sistemas de comunicación. La suite COMINT funciona con equipos para goniometría en bandas VHF/UHF y su generación de entornos goniométricos para su evaluación, visualización y análisis de señales recibidas, además de monitorización de comunicaciones tanto para móviles como satélites.

En consecuencia, surge la importancia de colocar estos diseños/prototipos bajo campañas de verificación para comprobar que el funcionamiento es óptimo en las condiciones reales donde se van a situar.

Debido al sector donde se mueven estos productos de COMINT, es necesario la búsqueda de soluciones basadas en el procesamiento de Señales Digitales como de Sistemas de Comunicación. Para ello, es necesario de un hardware específico que estimule las diferentes partes de un producto a través de sus entradas y, por la respuesta que devuelve, comprobar su validez.

Este TFM se centra en el sector de las radiofrecuencias, donde a partir de una placa de evaluación, se quiere estimular y analizar el comportamiento de los DACs y ADCs integrados a la hora de enviar/recibir una señal (o conjunto de ellas) sobre un producto prefabricado. Para conseguirlo, se ha necesitado de etapas de procesamiento (a través de filtros FIR de Interpolación y Diezmado) para alcanzar las particularidades de la señal buscada.

Por las especificaciones impuestas al diseño, se ha trabajado para la placa de evaluación ZCU111 (RFSoC Gen. 1). Aunque actualmente se tiene como objetivo su implementación en las siguientes generaciones. Principalmente la ZCU208 (RFSoC Gen. 3), la cual aprovecha mejor sus prestaciones disminuyendo la carga del PL (en aspectos de cómputo disminuyendo el uso de DSPs) y consiguiendo un mayor alcance en la tasa de muestreo (*Sampling Rate*).

Matizar la importancia de la programabilidad que se obtiene en el uso de estas placas de evaluación, ya que suponen un ahorro económico a nivel de empresa muy importante, el hecho de poder trabajar con circuitos integrados compuestos por interconexiones entre bloques de lógica con la posibilidad de ser reconfigurables. A diferencia de trabajar con ASIC o ASSP, donde el hardware es fijo y su posible modificación conlleva un alto coste tanto económico como de tiempo.

Actualmente, gracias a las ventajas que ofrecen las FPGAs, se ha permitido una mayor personalización de los sistemas electrónicos, adaptándolos a las necesidades particulares de un proyecto. Es por ello, que debido al crecimiento y constante evolución que está

ocurriendo en el campo de la electrónica, dicha adaptabilidad se convierte en una ventaja fundamental frente a las tecnologías actuales.

1.2 Objetivos

El objetivo fundamental de este proyecto ha sido la realización e implementación de un diseño para transmitir y recibir estímulos de cierto producto de COMINT relacionado con el campo de las radiofrecuencias.

Para cumplir con el objetivo propuesto, se han dictaminado varias especificaciones (desarrolladas en el Capítulo 4.1) a la hora de diseñar.

Para la realización de este proyecto, se ha tenido en cuenta la tasa de transferencia efectiva (*throughput*) y el ahorro de recursos utilizados (carga computacional utilizada) en la implementación del diseño.

Ya que se está trabajando en la creación del hardware mediante el entorno Vivado, este diseño depende principalmente de los *IP cores* que suministra Xilinx en su catálogo. En aquellos casos cuya funcionalidad no se ha podido obtener del repositorio, se ha realizado su creación a partir de *cores* sintetizados por el lenguaje de descripción VHDL.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- Capítulo 1: Introducción.
- Capítulo 2: Estado del Arte.
- Capítulo 3: Conceptos a tratar.
- Capítulo 4: Especificaciones y análisis.
- Capítulo 5: Integración del Hardware.
- Capítulo 6: Configuración del Software.
- Capítulo 7: Resultados.
- Capítulo 8: Conclusiones y trabajo futuro.

Para facilitar una lectura fluida, a lo largo de la memoria se ha decidido utilizar los nombres de terminología extranjera en su idioma origen, a partir del formato Italic o con comillas.

2 Estado del arte

2.1 FPGA

Una FPGA es un dispositivo basado en la configuración de sus bloques de lógica (LBs) para la implementación de un circuito integrado. Su realización tiene un proceso similar en todos los casos, donde el punto de salida es a partir de la descripción del comportamiento mediante un lenguaje de descripción de hardware de relativamente bajo nivel (HDL), o en su defecto por *cores* (bloques que contienen la lógica para realizar una funcionalidad) depositados en un catálogo, hasta su implementación en la placa.

Básicamente, una FPGA está compuesta por el conjunto de bloques lógicos configurables. Estos a su vez tienen como dos componentes principales FFs y LUTs. La combinación de estos componentes a partir de la matriz de interconexión permite implementar una funcionalidad lógica reflejado en tablas de verdad.

Además de estos componentes principales con sus bloques de entrada y salida (IOB) para la comunicación entre la FPGA y el mundo exterior, existen otros componentes básicos internamente: memorias RAM, multiplicadores (introducidos en la parte del silicio para el ahorro de CLBs) y gestores de reloj digital DCM para proporcionar sincronismo a la hora de generar relojes.

En el mercado de este sector, las FPGAs han sido enormemente utilizadas en aplicaciones embebidas para la adquisición de datos reales, filtrados y procesamiento de señal. Por ello, gracias a las capacidades que contiene la RFSoC de Xilinx, se ha podido desplegar todo este proyecto. La RFSoC surge de la familia de Sistemas "Zynq" para SoC (*System-on-a-chip*), cuyo objetivo viene directamente para el campo de las Radiofrecuencias.

Estos dispositivos (de igual manera a la arquitectura de la familia Zynq-7000) están divididos internamente en dos bloques principales. El PS ("Processing System"), es decir, el sistema de procesamiento y el PL ("Programming Logic"), la lógica programable. Las funciones principales de estas dos subdivisiones son: por parte del PS la integración de la funcionalidad buscada y por parte del PL, la implementación de la lógica (a través del hardware).

Esta forma de dividir la arquitectura en dos apartados permite alcanzar niveles de actuación más elevados que lo conseguido con un gran número de chips (ya sea por aspectos de latencia, ancho de banda del procesador, etc.). Además, que el uso del PL traslada tareas del PS, acelerando y mejorando su rendimiento. Otra ventaja que permite este aislamiento latente es que posibilita reducir el consumo de energía operando exclusivamente a partir del PS por sus características de "*hard-core*", facilitando trabajar independientemente del soporte de una FPGA.

Por ello, antes de continuar detallando las características de las placas de evaluación utilizadas, es esencial empezar mencionando el protocolo AXI: La estructura básica que conforma los buses de comunicación para las arquitecturas basadas en Zynq.

2.2 Protocolo AXI

El protocolo AXI (*Advanced extensible Interface*) surge con el objetivo para la comunicación entre los módulos implementados en una placa de evaluación. Se trata de una tecnología inculcada en los *buses-on-chip*.

Los *buses on-chip* comienzan siendo una alternativa como arquitectura estandarizada de las interfaces de comunicación (entre *cores*). Esto permite aportar diversas ventajas a la hora de operar, como la simplificación de librerías a partir de componentes comunes, la facilidad del proceso de integración y verificación, y la mejora, en cuestión del diseño, a partir de prestaciones más complejas (consiguiendo mayor eficiencia).

Con esto surge la especificación de AMBA en 1996.

2.2.1 AMBA

El término AMBA (*Advance Microcontroller Bus Architecture*) es el protocolo agenciado para la interconexión entre componentes dentro de las arquitecturas de procesadores ARM. Actualmente se ha convertido en el estándar principal para los diseños SoC (*System-on-chip*).

La evolución que ha sufrido a lo largo de los años ha determinado diferentes versiones en su estructura. Sus inicios introducen al mercado los denominados buses ASB (*Advance System Bus*) y APB (*Advanced Peripheral Bus*), siendo estos la primera versión comercial. A lo largo de los años van adquiriendo mayor complejidad aportando dispositivos de alta frecuencia.

2.2.2 AMBA AXI4

El protocolo AXI4 es una actualización del protocolo ACE (*AXI Coherency Extensions*) que añadió 3 canales para compartir datos entre la comunicación de cachés de tipo “maestro” y un hardware de control y mantenimiento, además de la posibilidad de mantener pendientes transacciones múltiples. Esto conlleva una mejora en el rendimiento, además de posibilitar el uso de interconexión entre varios interfaces “maestros”.

Las principales ventajas son:

- Proporcionar en una transacción ráfagas de alto rendimiento de hasta 256 transferencias para una única dirección.
- Generar un sistema de señalización para proporcionar mayor calidad de servicio.
- Incluir mayor diversidad de interfaces tanto de formato como de número.

El protocolo AXI proporciona conexiones tanto de direcciones como de datos independientes, aportando el formato “bidireccional”. Su estructura se define en 5 canales, dependiendo de la función realizada, el número de canales utilizados cambia.

Transacción de lectura

La transacción de lectura utiliza 2 canales, uno para la dirección y otro para los datos. El proceso comienza por parte de la interfaz “maestra” concretando la dirección y estableciendo algunas señales de control.

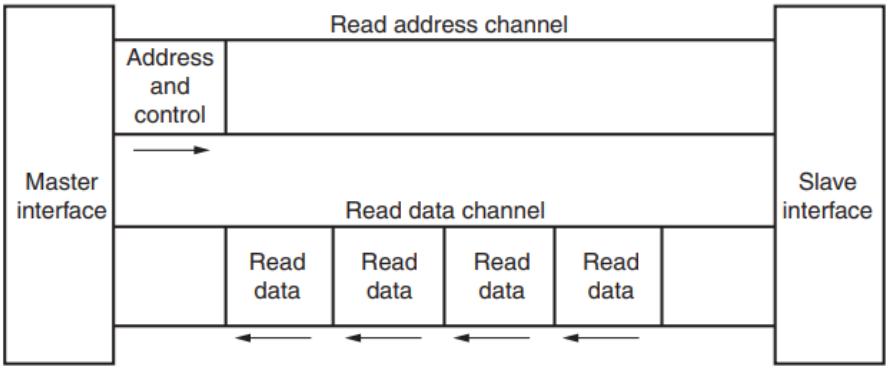

Figura 2-1: Transacción de lectura (AXI4) [1]

Transacción de escritura

La escritura, en cambio, se realiza mediante 3 canales: dirección, datos y respuesta. En este caso, no es necesario que comience la transacción aportando antes la dirección que los datos, lo cual proporciona mayor agilidad en la transferencia.

A partir de este protocolo, se puede substraer tres interfaces: AXI4-Memory Mapped (Full AXI), AXI4-Lite y AXI4-Stream.

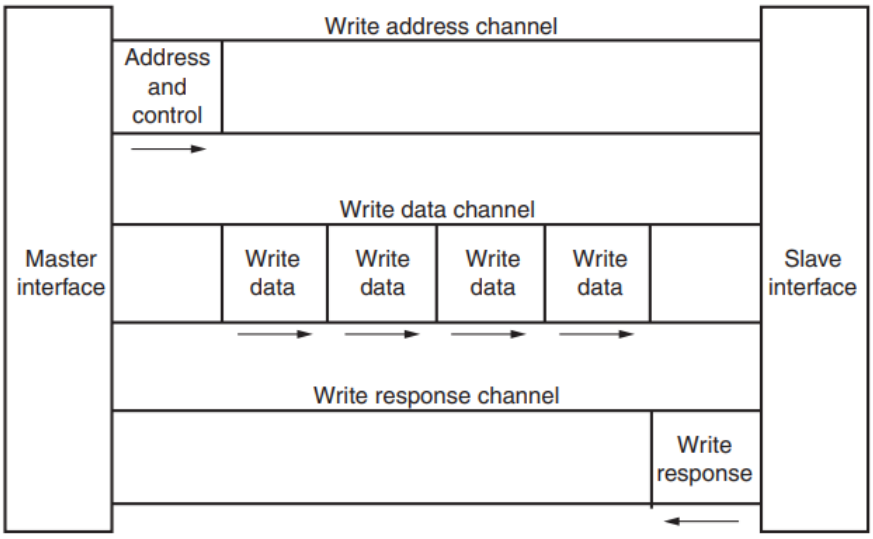

Figura 2-2: Transacción de escritura (AXI4) [2]

2.2.2.1 AMBA AXI4-Memory Mapped

El protocolo AXI4-Memory Mapped tiene como objetivo la asignación de memoria para alto rendimiento. El ancho de los datos puede ser de 32, 64, 128, 256, 512 o 1024 bits.

A diferencia del siguiente protocolo presentado AXI4-Lite, que permite una única transferencia de datos por transacción, AXI4-Memory Mapped limita hasta un máximo de 256 transferencias. Establecer este límite aporta la mayor tasa de transferencia efectiva disponible (objetivo buscado en este proyecto).

2.2.2.2 AMBA AXI4-Lite

El protocolo AXI4-Lite va orientado a la comunicación con interfaces de formato de registro de control para niveles más simples. Las funcionalidades principales que aporta es que todas las transacciones tienen una longitud de una ráfaga, el tamaño de los accesos de datos es equitativo al ancho de bus de datos (típicamente 32-64 bits) y no permite el acceso exclusivo (además de no ser modificable).

La utilización de este protocolo es únicamente para la configuración de registros, lo cual permite que este tipo de transferencias no necesite una alta velocidad para realizarse.

Las señales que este protocolo proporciona son las siguientes:

Global	Write address channel	Write data channel	Write response channel	Read address channel	Read data channel
ACLK	AWVALID	WVALID	BVALID	ARVALID	RVALID
ARESETn	AWREADY	WREADY	BREADY	ARREADY	RREADY
–	AWADDR	WDATA	BRESP	ARADDR	RDATA
–	AWPROT	WSTRB	–	ARPROT	RRESP

Figura 2-3: Señales del protocolo AXI4-Lite [3]

2.2.2.3 AMBA AXI4-Stream

El protocolo AXI4-Stream, también denominado “AXIS”, se utiliza como interfaz estándar en la conexión de componentes cuyo objetivo es el intercambio de datos. La interfaz recae como mínimo en un “maestro” que genera datos, los cuales son recibidos por un “esclavo”. Este protocolo admite múltiples flujos, lo cual permite compartir la transmisión por los mismos buses, combinar la interconexión genérica y realizar operaciones de ampliación, reducción y enrutamiento.

Este protocolo admite una gran diversidad de formatos de transición, lo cual se encarga de asociar la relación transferencia-paquetes en el flujo de datos.

Los tipos de *stream* de datos vienen establecidos por la estructura ejercida por el protocolo AXI4-Stream. Aunque la distribución interna varíe la información, sigue permaneciendo intacta. La estructura viene determinada por la serie de datos que comprende la información y los *null bytes.*

Null	Null	D-07	D-0A	Null	D-0F
D-01	Null	D-06	D-09	Null	D-0E
Null	D-03	D-05	D-08	D-0C	Null
D-00	D-02	D-04	Null	D-0B	D-0D

D-02	D-06	Null	D-0B	Null	Null
Null	D-05	Null	D-0A	D-0E	D-0F
D-01	D-04	Null	D-09	D-0D	Null
D-00	D-03	D-07	D-08	D-0C	Null

Figura 2-4: Dos ejemplos de *stream* (misma información) [4]

Dependiendo de la existencia de estos *null bytes* en el *stream,* aparecen tres tipos:

- *Stream* alineado continuo: Es la transmisión de un conjunto de datos donde no aparecen *null bytes*.
- *Stream* desalineado continuo: Aquella transferencia donde aparecen *null bytes* en el principio, en el final o en ambas partes.
- *Stream "*disperso*"*: Cuando la transferencia no es un caso anterior y contiene una distribución variable que debe mantenerse entre bytes de datos y *null bytes*. En comparación, suele existir un mayor número de bytes de datos.

Lista de señales del protocolo AXIS

La lista de señales de este protocolo son las siguientes:

- **Aclk:** Es la señal de reloj principal. La confirmación de todas las señales restantes se lleva a cabo en el flanco de subida de esta señal.
- **Aresetn:** La señal de reset global. Funciona en nivel "activo-bajo".
- **Tready:** Esta señal revela a la interfaz "esclava" que puede aceptar dicha transferencia en el ciclo actual en donde se encuentra.
- **Tvalid:** Es la señal que indica que el dato de la transferencia proveniente del "maestro" es válido. Sin embargo, para que haga efecto la transmisión, es requerida la confirmación de la anterior señal, el *tready.*
- **Tlast:** Es una señal que se activa a "1" cuando se encuentra en el ciclo final del *stream*.
- **Tdata:** Es el término "*payload*", proporciona la información base. Su formato viene determinado por un número entero de bytes.
- **Tkeep:** Sirve como calificador de bytes, determina qué cantidad de bytes son nulos y, por tanto, eliminables del *stream.*
- **Tstrb:** Esta señal determina si el contenido del *tdata* es procesado como un byte de datos o un byte de posición.
- **Tid:** Es el identificador para diferentes *stream*.
- **Tdest:** Esta señal suministra información en el enrutamiento del *stream.*
- **Tuser:** Es una señal definida por el usuario que indica el inicio del *stream.*

Sin embargo, para realizar diseños óptimos con la interfaz "AXIS", se debe enfocar en las siguientes señales: *tdata, tvalid, tready* y *tlast.* Aunque es evidente, se está obviando la imprescindibilidad de las señales *aclk* y *aresetn* para el funcionamiento del diseño**.**

Para conseguir un diseño con las mejores garantías se ha introducido el modo "Blocking Type" o "control de flujo", es decir, la utilización de la señal *tready + tvalid* en todo momento. Con la aportación de la señal *tready* se asegura que en la transmisión no existe ningún tipo de pérdida de datos, añadiendo la señal *tvalid* se obtiene su veracidad.

A continuación, se muestran más detalles de esto último, aportando tanto ventajas como desventajas en el uso de la misma.

2.2.2.3.1 AXIS con modo "Non-Blocking"

El modo "Non-Blocking" de AXIS, es la interfaz en la que trabaja únicamente con la señalización del *tvalid*. Esta señal es el "controlador" más típico utilizado en una transmisión, su mayor ventaja es la simpleza en su fabricación (para ello, solo es necesario conocer el conteo de datos).

El problema surge cuando la transmisión de datos no es regular, apareciendo la pérdida de datos (sobreescritura sobre los datos antiguos) debido al desconocimiento que tiene la interfaz del "maestro" sobre la situación actual del "esclavo".

En la Figura 2-5, se muestra un ejemplo donde sucede esta complicación. Se trata de un diseño donde tiene implementado un contador de 16 bits, el cual está conectado a una FIFO (módulo de almacenamiento) de 16 posiciones de profundidad.

La etapa A muestra la transmisión normal de datos. Como no existen "complicaciones", la FIFO no requiere que almacene datos (ya que existe una transmisión completa, es decir, una palabra por entrada y otra por salida).

Esto sucede hasta el punto B, donde a la FIFO se le incapacita la emisión de datos (corte del flujo de datos). Esto ocasiona que su buffer interno comience a llenarse, ya que el otro módulo, el contador, sigue funcionando (por el desconocimiento del estado de la FIFO). Existe un punto (etapa C) donde el límite de almacenamiento se excede, produciendo la pérdida de datos.

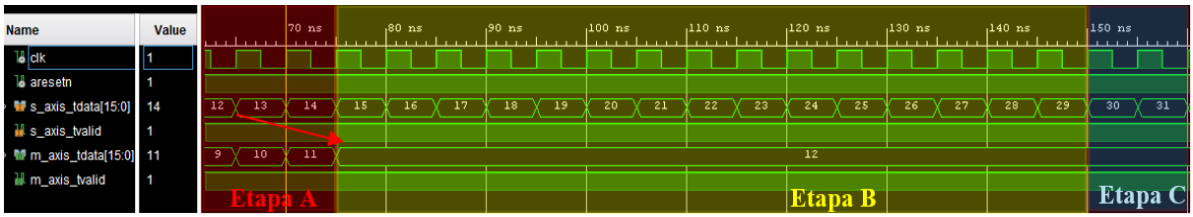

Figura 2-5: Ejemplo con modo "Non-Blocking", parte 1

Ciclos más adelante se vuelve a establecer el proceso (transición de la etapa C a la etapa A). Sin embargo, una serie de datos se han perdido por la sobreescritura.

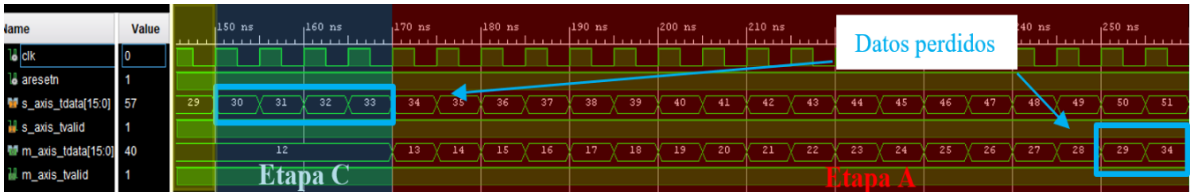

Figura 2-6: Ejemplo con modo "Non-Blocking", parte 2

2.2.2.3.2 AXIS con modo "Blocking"

Para este modo están presente tanto las señales *tvalid* como *tready*. Esto determina cuando una transmisión se puede realizar.

Gracias a esta señalización, existe el conocimiento de la situación actual entre las dos interfaces ("esclavo" y "maestro"). Esto se conoce como "Control de Flujo" (*Flow Control*), y hace referencia a la posibilidad de parar o solicitar datos cuando sea necesario.

Si se introduce esta funcionalidad en el ejemplo anterior, la comunicación del control de flujo entre el "esclavo" y el "maestro" existe cuando el buffer de la FIFO está completo. De tal manera que la señal del *tready* del "esclavo" detiene el funcionamiento del contador hasta que el flujo se reanuda.

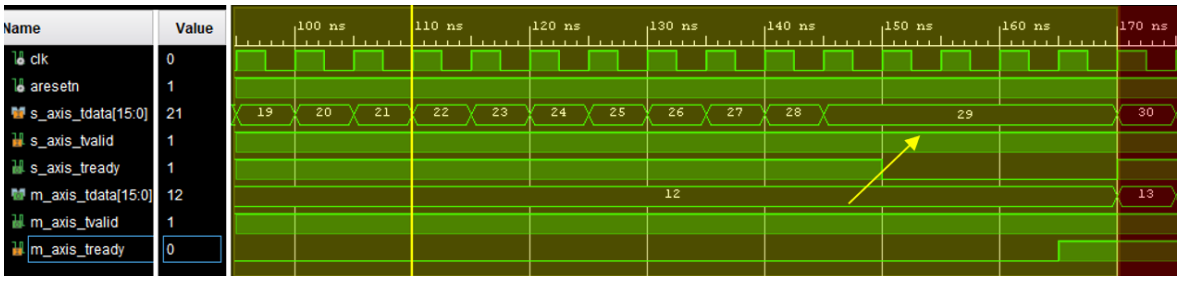

Figura 2-7: Ejemplo con modo "Blocking"

Este autoabastecimiento es clave para el funcionamiento del diseño implementado en este proyecto. La utilización de FIFO con control de flujo adecúan la transmisión de datos a distintas velocidades (frecuencias).

Las señales *tready* y *tvalid* establecen cuándo se valida la transmisión de información a través de la interfaz. Es un mecanismo en modo bidireccional (tanto el "maestro" como el "esclavo" controlan el proceso). La validación de ambas señales se puede realizar o no en el mismo flanco de reloj.

A partir de esto, aparece una serie de consideraciones, que determinan cuando se ha establecido la comunicación entre interfaces. La validación de esta comunicación se manifiesta como "*handshake*".

A continuación, se muestra esta serie de consideraciones:

- En la interfaz del "maestro" es necesario que el *tvalid* esté activo para poder llegar a confirmar el *tready* del "esclavo" . Este proceso debe permanacer en el tiempo hasta que se complete el "*handshake*".

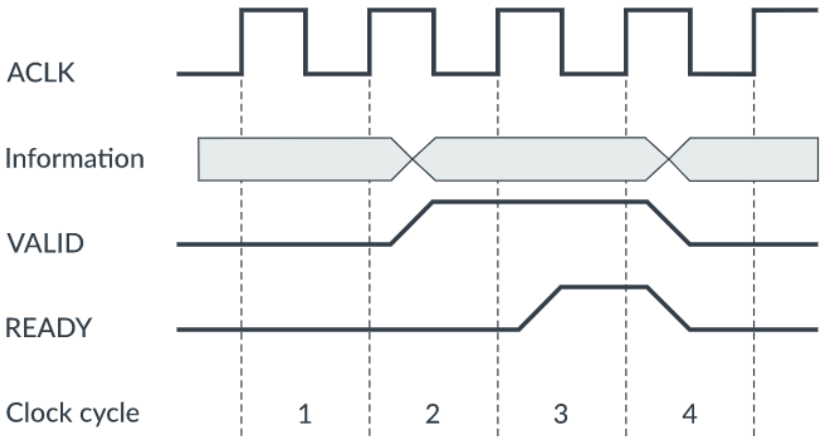

Figura 2-8: Ejemplo *"handshake" 1* [5]

- Para que la transmisión se pueda realizar en un único ciclo de reloj, el *tready* del "esclavo" tiene que estar activo previamente a la llegada de los datos y lo mismo del *tvalid* por parte del "maestro".

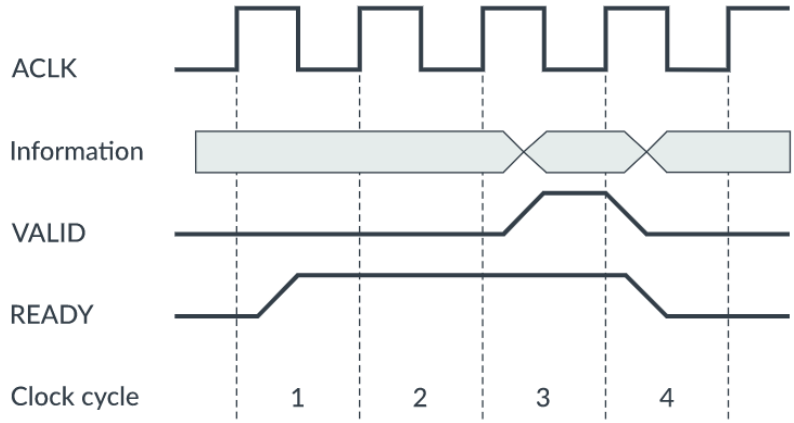

Figura 2-9: Ejemplo "*handshake*" 2 [6]

- También puede suceder el caso anterior donde el *tvalid* del "maestro" y el *tready* del "esclavo" se activan en el mismo flanco de reloj.

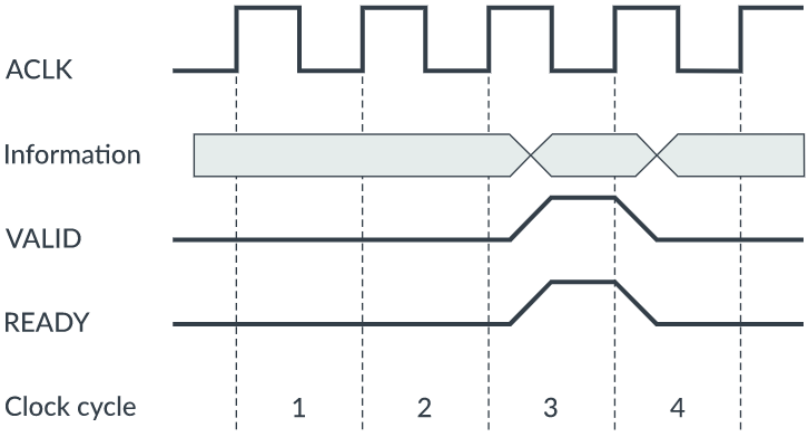

Figura 2-10: Ejemplo "*handshake*" 3 [7]

Teniendo en cuenta lo anterior, este proyecto evita el estancamiento del flujo de datos a partir de la "encolación" de los datos válidos. Siempre y cuando estén disponibles. Esto ahorra la comprobación previa si el *tready* se encuentra válido.

2.3 Entorno de desarrollo

El entorno de desarrollo permite toda la metodología de implementación de circuitos digitales sobre FPGAs.

En este proyecto se ha utilizado *Vivado Design Suite*, con versión 2019.1. Este entorno es compatible con las siguientes familias de FPGAs: *Artix*, *Virtex*, *Kintex*, *Spartan-7*, *Zynq* y *UltraScale+* (*MPSoC* y *RFSoC*). Siendo esta última la utilizada en este trabajo.
Este proyecto cumple con un diseño basado en la utilización de la herramienta *IP Integrator*. Dicha herramienta introducida en el entorno permite la creación de diseños a partir de una interfaz visual. El procedimiento comienza con la selección de *IP Cores* de un repositorio, la modificación de su configuración a partir de una API y la realización del plano de uniones para su interacción.

2.4 Metodología para la implementación de un circuito

En este apartado se detallan las diferentes etapas para la implementación del diseño.

- **Creación del diseño a partir de *IP Integrator*:** El diseño realizado se integra con *IP Cores* obtenidos a partir del catálogo que suministra Vivado (Xilinx). En el caso que la funcionalidad requerida a partir de un módulo no pueda obtenerse del repositorio, se realiza su implementación mediante lenguaje de descripción de circuitos (VHDL).

- **Simulación**: Una manera de comprobar que las diferentes partes del diseño funcionan de la manera correcta es mediante esta etapa. Para ello, se abstrae la parte del diseño que se quiere comprobar (la disminución de un diseño de *testing* agiliza enormemente los tiempos de simulación). En Vivado, la herramienta por defecto utilizada y que se ha manejado es: "Vivado Simulator".

- **Proceso de "*constraint*" del circuito**: Las restricciones físicas a las que está sujeto el diseño se dividen principalmente en dos apartados: el *placement* y el *timing*. El *placement* se basa en la integración de los pines requeridos (además de sus propiedades), haciendo que el diseño finalmente se encuentre implementado con los componentes físicos conectados. El *timing* se utiliza principalmente en introducir los detalles para aspectos de frecuencia.
  En este último, se obtiene la generación de relojes a partir de sintetizadores externos a la RFSoC (el LMK y el LMX). Para programar este archivo de *constrain*, es necesario tener en cuenta la documentación de Xilinx de la placa de evaluación.
  Respecto al archivo XDC estándar, se puede encontrar en la página oficial de Xilinx, localizado en la sección de la ZCU111 (apartado "Board Files" [28]).

- **Proceso de Síntesis**: Es la etapa que convierte la descripción de un diseño basado en RTL a un circuito físico. Para ello, es necesario que las diferentes condiciones que afectan al circuito se puedan implementar. El proceso realizado se evalúa a nivel de cada módulo del diseño. El término síntesis hace referencia usualmente al proceso de conversión de texto a puertas lógicas o multiplexores.

- **Proceso de Implementación**: El objetivo de esta etapa es la conversión del diseño sintetizado a un formato compatible con los recursos físicos de la FPGA. En este proceso, se comprueba si las *constrain* aplicadas se pueden integrar.

- **Proceso de Programación y Debug**: Es la etapa que realiza el volcado de valores que implementan el diseño en la FPGA (*bitstream*).

Como etapa extra, aparece:

- **Proceso de Software**: Se basa en la conversión del *bitstream* al apartado software, él se encarga de implementar todo el diseño en la placa de evaluación. También se incluye la integración de la funcionalidad buscada. Este trabajo se basa principalmente en trabajar con la plataforma PYNQ.

2.5 Placa utilizada

En la realización de este trabajo se ha utilizado la placa de evaluación "Zynq® UltraScale+™ RFSoC ZCU111", más conocida comúnmente por su orden de generación: "RFSoC Gen. 1".

Figura 2-11: Placa de evaluación ZCU111 (RFSoC) [8]

Debido al uso intensivo que sufre esta placa en la empresa SENER, se han experimentado pruebas intermedias de implementación (de carácter menor) con la "Zynq UltraScale+ MPSoC ZCU102".

Figura 2-12: Placa de evaluación ZCU102 (MPSoC) [9]

2.5.1 Características

La placa de evaluación RFSoC ZCU111 (igual que la MPSoC ZCU102) forman parte de la familia Zynq de Xilinx sobre *SoC* de gama alta. Usan la arquitectura *UltraScale+* evolucionada de la conocida arquitectura conocida como "7-series".

2.5.1.1 Sistema de Procesamiento (PS)

A continuación, se comparan las prestaciones referidas al sistema de procesamiento (PS).

	Zynq UltraScale+ MPSoC	Zynq UltraScale+ RFSoC
Dispositivo Ultrascale+ (SoC de la Zynq)	ZU9EG (XCZU9EG-2FFVB1156E)	ZU28DR
Unidad de procesamiento para aplicaciones	ARM Cortex-A53 (4 núcleos MPCore) hasta Frec. máx. de 1.5 GHz	ARM Cortex-A53 (4 núcleos MPCore). Frec. máx. de 1.3 GHz
Unidad de procesamiento a tiempo real	ARM Cortex-R5 (2 núcleos MPCore) Frec. Máx: 600 MHz	ARM Cortex-R5 (2 núcleos MPCore) Frec. Máx: 533 MHz
Interfaz de memoria dinámica	DDR3, DDR3L, LPDDR3, DDR4 (4GB, 64b y 2666MT/s), LPDDR4	
Conectividad (periféricos de alta velocidad)	USB 3.0, Serial ATA 3.1, DisplayPort, Ethernet (Gigabit), SD/SDIO	
Conectividad (periféricos de baja velocidad)	USB 2.0, UART, CAN, I2C, bus SPI, 32b GPIO	
Pines I/O	214	
Seguridad	RSA, AES, SHA y ARM TrustZone	

Tabla 1: Tabla de características del PS (RFSoC vs. MPSoC)

Se puede observar que muchas de las prestaciones que tienen ambos dispositivos por parte del PS son muy similares. Un punto destacable es que la RFSoC, a diferencia de la MPSoC, opera a una frecuencia más baja (en torno a un 10 % menor). Además, que esta última contiene para el apartado de procesamiento multimedia una GPU ARM $Mali^{TM}$-400 MP2 con frecuencia máxima hasta 667 MHz.

Si se comparan los diagramas de bloques de ambas placas, se observa la diferencia intrascendente que existe. La Figura 2-13 expone el diagrama de la MPSoC extraído del documento de "User Board" de Xilinx. Si se elimina la parte del recuadro subrayado se obtiene el equivalente a la RFSoC.

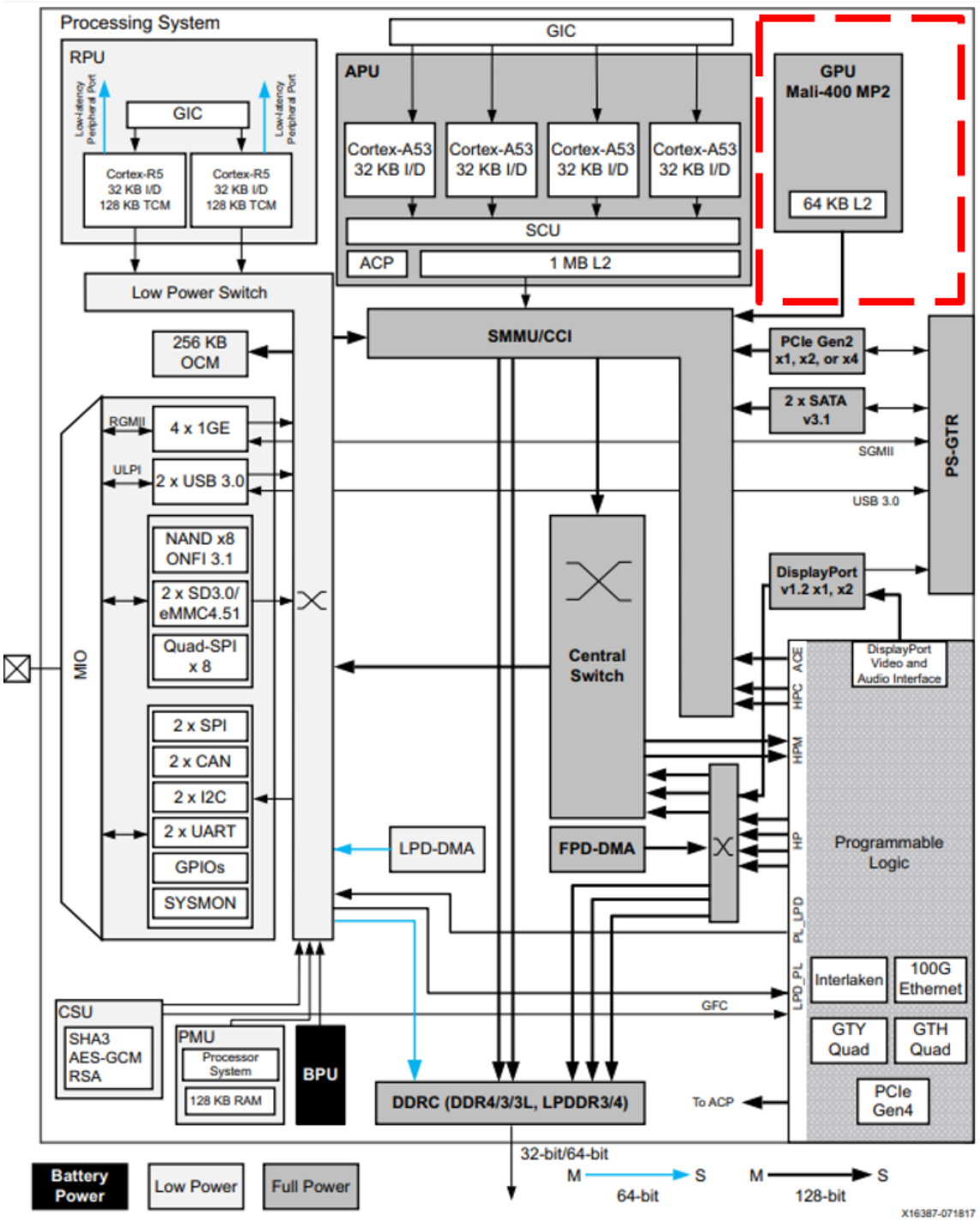

Figura 2-13: Diagrama de bloques RFSoC y MPSoC [10]

2.5.1.2 Lógica programable (PL)

En este apartado es donde aparece la mayor diferencia. Tan solo con echar un simple vistazo a la diferencia de tamaño en ambas tarjetas es suficiente (RFSoC: 600 cm^2 vs. MPSoC: 500 cm^2).

	Zynq UltraScale+ MPSoC	Zynq UltraScale+ RFSoC
Celdas lógicas [K]	600	930
Memoria máxima [Mb]	32.1	60.5

Transceptores GTH (16 Gbps)	24	-
Transceptores GTY (33 Gbps)	-	16
Sectores DSP	2520	4272
Pines I/O	328	347

Tabla 2: Tabla de características del PL (RFSoC vs. MPSoC)

En este apartado del PL, la RFSoC contiene mejores prestaciones. Si se realiza la comparación con la MPSoC, integra el doble de memoria, un mayor número de DSPs y, respecto a celdas lógicas, contiene un 33% más. No posee transceptores GTH, pero posee transceptores GTY, los cuales operan al doble de velocidad.

Sin embargo, lo más determinante y con relación a este proyecto, es la incorporación de una cadena de radiofrecuencia, donde la MPSoC carece de ella.

	Zynq UltraScale+ MPSoC	Zynq UltraScale+ RFSoC
RF-ADC	-	8 [12-bit, 4.096 GSPS]
RF-DAC	-	8 [14-bit, 6.554 GSPS]

Tabla 3: Características de la cadena de radiofrecuencia (RFSoC)

3 Conceptos

En este apartado se van a analizar los conceptos claves en la realización del diseño, que vienen en relación por los requerimientos previos establecidos (descritos en el apartado 4.1). Estos requerimientos dictaminan que el diseño implementado consigue el comportamiento buscado.

Este apartado tiene dos conceptos principales: la tasa de muestreo (*Sampling Rate*) y la tasa de transferencia efectiva (*throughput*).

3.1 Tasa de muestreo

La tasa de muestreo o también por su denominación anglosajona *Sampling Rate*, es la cantidad de muestras obtenidas de una señal por intervalo de tiempo para obtener de una señal continua su análoga en el dominio discreto (A/D) o el dominio continuo (D/A).
Esta conversión permite a los ordenadores/computadoras poder interactuar con señales analógicas. Sin embargo, esta conversión repercute en la calidad de información resultante. Por ello, es necesario comprender qué parte de la información es básica para que después del proceso permanezca intacta.

Para esta conversión se ha de tener en cuenta tres aspectos: el muestreo derivado por el Teorema de Nyquist, la modificación de la frecuencia de muestreo y la cuantificación (la resolución en bits).

3.1.1 Teorema de *Nyquist*

Para realizar un muestreo adecuado se debe seguir el Teorema de Nyquist-Shannon. Este teorema determina los detalles para realizar una conversión de analógico a digital y viceversa, para grandes distancias y obtener un resultado fiable. Para ello, es necesario que la señal se muestree al doble de la frecuencia máxima elegida.
En la práctica, la no existencia de filtros paso-bajo analógicos ideales, obliga a dejar un cierto margen entre la frecuencia máxima y la frecuencia de Nyquist.

Esta frecuencia crítica (Nyquist) es:

$$f_e < \frac{f_s}{2}$$

Ecuación 1: Ecuación Teorema de Nyquist [37]

Siendo f_e la frecuencia seleccionada y $\frac{f_s}{2}$ la frecuencia crítica o de Nyquist. La frecuencia de muestreo es:

$$2 \times f_e = f_s$$

Ecuación 2: Ecuación de frecuencia de muestreo

No seguir la norma que rige Nyquist puede originar la aparición del fenómeno *aliasing*, es decir, la superposición sucesiva de réplicas de la señal transmitida. Con ello, se obtiene una mala reconstrucción de la señal al convertirse en un proceso indistinguible en el filtrado.

En la Figura 3-1, se puede observar que la frecuencia de muestreo realizada no es suficiente para su reconstrucción.

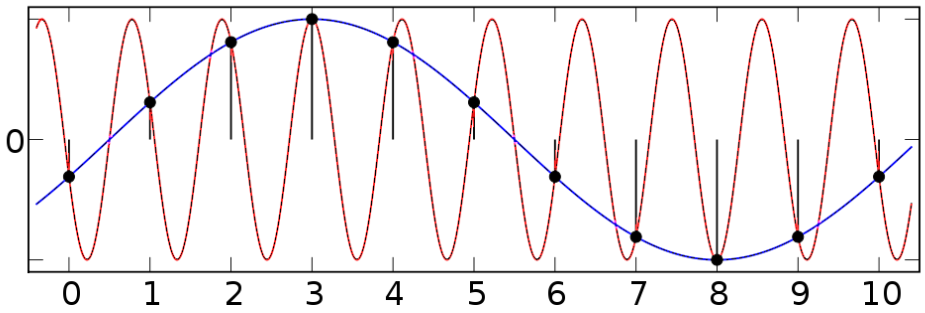

Figura 3-1: Representación de un muestreo escaso [11]

3.1.2 Modificación de la frecuencia de muestreo original

Otro punto que afecta al aspecto del muestreo es la generación de la señal en función de su frecuencia. Ya que en este proyecto se pretende obtener una tasa de muestreo en concreto, es necesario realizar la conversión de la frecuencia en el dominio digital.

Los procesos principales son: La interpolación y el diezmado.

Antes de realizar una breve descripción de estos procesos, hay que comentar que tanto la interpolación como el diezmado se procesan en el diseño a partir del módulo FIR Compiler y el Zynq Ultrascale+ RF Data Converter (a través de sus DUCs y DDCs). La forma de operar depende del grado de *upsampling/downsampling* que se esté realizando.

Es interesante conocer que cuando se está trabajando con un alto grado, la manera más apropiada de realizar dichas operaciones es dividiendo el proceso en varias etapas (en modo "cascada"). Esto reduce enormemente la carga computacional, algo que en las FPGAs es limitado.

En la Figura 3-2, se describe el proceso que se realiza en el módulo Zynq Ultrascale+ RF Data Converter para escoger el factor de *upsampling/downsampling* a realizar. En este caso, las opciones son estándares (1, 2, 4 u 8) y la manera de elegirlo es a partir de un multiplexor donde cada opción viene determinada por su entrada de control.

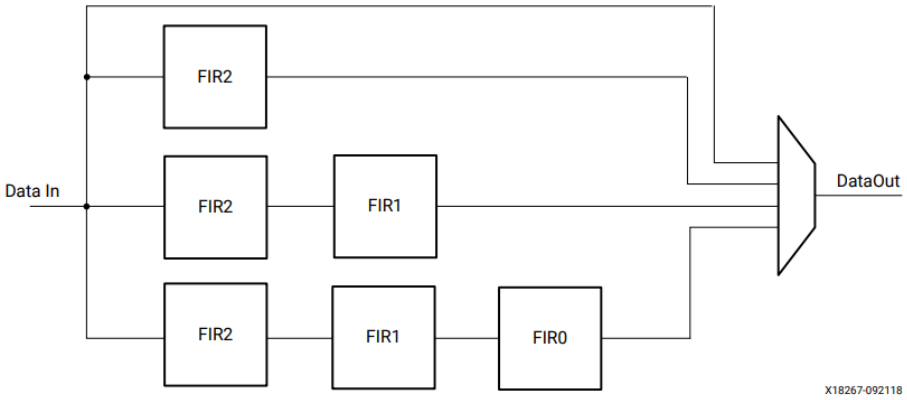

Figura 3-2: Estructura de *upsampling/ downsampling* del RFDC [12]

3.1.2.1 Interpolación

La interpolación es el proceso de empleo común para realizar la reconstrucción "aproximada" o exacta de una señal a partir de sus muestras. La situación más ideal se obtiene cuando los instantes de muestreo están lo suficientemente cerca para hacer una reconstrucción idéntica a la original.

Aunque existen muchas opciones para realizar esta operación: splines, cuadrática… (basadas en introducir información "inventada"), se opta por el *upsampling*, la cual preserva en la señal su contenido espectral.

El proceso de *upsampling* se basa en dos pasos: la expansión y la interpolación. Este primer paso introduce entre los diferentes puntos de la señal una serie de ceros, cuyo número viene referido al grado de interpolación escogido. El segundo paso incorpora el filtro de interpolación donde se suavizan las discontinuidades (surgidas por la sucesión de ceros) por la incorporación de valores interpolados (promedio entre puntos).

Sin embargo, este segundo paso produce la aparición sucesiva de réplicas de la señal original. Para evitarlo, se introduce esta señal interpolada por un filtro de paso bajo específico a la frecuencia.

En la Figura 3-3, se muestra el resultado de este proceso tanto en el dominio del tiempo como el de la frecuencia para una interpolación con factor L=3.

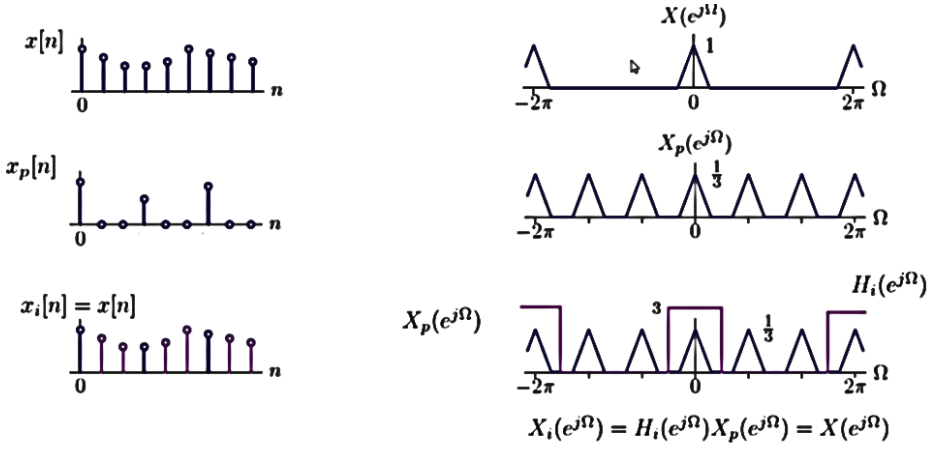

Figura 3-3: Proceso de Interpolación [13]

Respecto al efecto que produce a la frecuencia de muestreo, la interpolación aumenta dicha frecuencia un factor L.

3.1.2.2 Diezmado

El diezmado o *downsampling*, es la otra operación para la modificación de la frecuencia de muestreo. En este caso, la señal resultante contiene una frecuencia de muestreo reducida un factor M.

El procedimiento se realiza en dos pasos: el primer paso es la inserción previa de un filtro paso bajo adecuado a la frecuencia de corte para la eliminación del "*aliasing"* y el segundo

paso, conlleva el diezmado de un factor M de la señal. Esto permite mantener M-enésimas muestras de la señal original.

En la Figura 3-4, se muestra el resultado en ambos dominios para un diezmado de M=3.

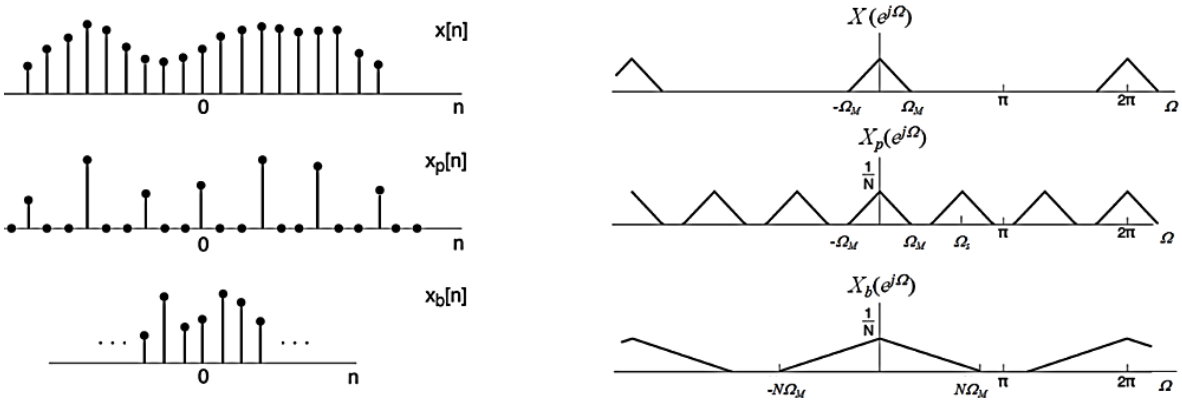

Figura 3-4:Proceso de Diezmado [14]

3.1.3 Cuantificación y resolución

La cuantificación es una etapa básica en el proceso de conversión digital, en ella, se asigna el nivel adquirido para su resolución en bits. Dependiendo de la conversión, los valores se establecen de forma distinta, no es equivalente el valor analógico de la señal con su equivalente en la señal digital. Es aquí donde aparece el error de cuantificación.

Otro concepto con relación a este anterior es la resolución en bits. Este concepto es la profundidad en bits que se necesita para muestrear una señal. En este diseño, la resolución viene afectada por la comunicación de los módulos a través de sus buses (donde el ancho viene correspondido por valores de potencia de dos).

Aunque más adelante se especifica, este proyecto trabaja con señales complejas de 32 bits. Donde en la parte menos significativa (LSB) establecida por 16 bits se localiza la parte real de la muestra (comúnmente declarada como "I") y la parte más significativa de 16 bits se representa la parte imaginaria ("Q").

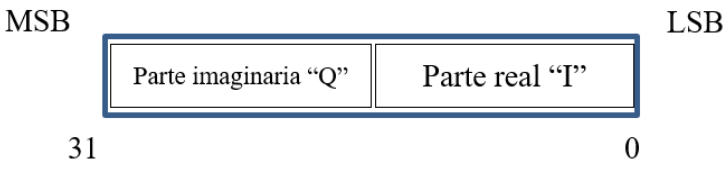

Figura 3-5: Formato de una muestra

En el procesamiento, la señal es introducida por diferentes etapas provocando cambios en sus valores. Por ello, es necesario conocer las posibles saturaciones o pérdidas que puedan acarrear en el proceso. Es común el uso de bloques de truncamiento donde depende del usuario la adaptación de la señal. Dicha señal se toma tras un desplazamiento lógico de la parte donde se considera que existe la mayor carga de información y, si existe saturación, que sus valores excedidos equivalgan al valor máximo posible.

Ya que el formato en Vivado se realiza en complemento a 2, que a diferencia de otros programas donde la saturación establecida para aquellos valores saturados equivale al máximo valor alcanzado por el tamaño de la palabra. En este caso, dichos valores son solapados en la siguiente escala, obteniendo un comportamiento descontrolado e inesperado.

En la Figura 3-6 se muestra este comportamiento, siendo la línea azul la señal original y la naranja la señal amplificada con saturación desfasada 180 grados (en función de la amplitud frente al tiempo).

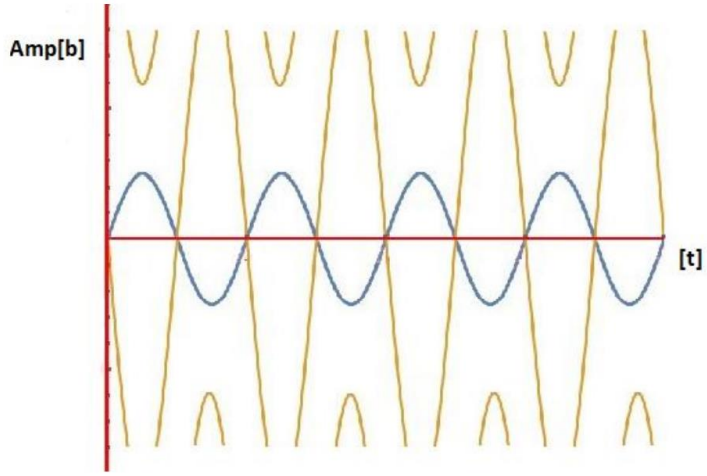

Figura 3-6: Saturación de una señal en el entorno Vivado

3.2 Tasa de transferencia efectiva

La tasa de transferencia efectiva o *throughput* es la cantidad de información neta que se transfiere a lo largo de un proceso. Por tanto, viene en relación directa a la cantidad de tiempo en realizarlo.

Como después se observa a lo largo de las etapas del diseño, la transferencia de datos para cierto ancho de bits influye en la frecuencia a la que trabajan los módulos, estableciendo el "caudal" por donde se procesan los datos (ancho de banda).

Para este diseño, existen dos modalidades para analizar: Transmisión y Recepción. Para el primero, el intervalo de tiempo consta desde que se transmiten los datos (originados en el PS y transferidos al PL) hasta que son mandados por la cadena de RF. Para el segundo, al sentido inverso, una vez introducidos los datos en la cadena de RF hasta su escritura en el PS.

El tiempo más determinante es el necesario que realizan los módulos AXI Memory Direct Access (DMA) en realizar los accesos en memoria. Esto es lo que conforma la comunicación PS-PL.

Esta tecnología tiene un comportamiento como se muestra en la Figura 3-7. Se trata de una gráfica que muestra el tiempo necesario para asignar cierto volumen de datos.

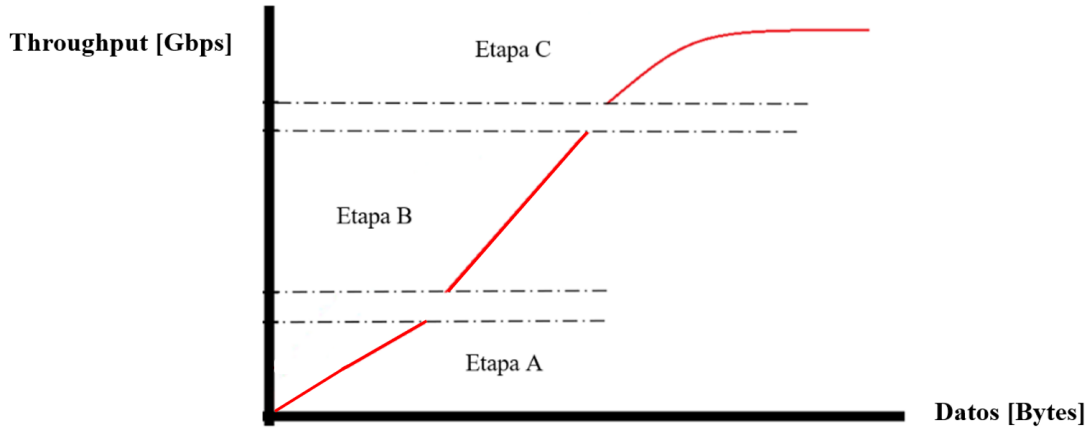

Figura 3-7: Comportamiento de la tasa de transferencia efectiva (*throughput)*

Analizando la gráfica, frente al incremento del volumen de datos a transmitir, el comportamiento de la tasa de transferencia efectiva adquiere 3 tendencias:

- **Etapa A**: Da igual el volumen de datos, al ser relativamente pequeño, el tiempo para la transición permanece constante (depende exclusivamente del acceso en memoria).
- **Etapa B**: El comportamiento para una cantidad de datos es directamente proporcional al intervalo de tiempo para su transmisión.
- **Etapa C**: La tasa de transferencia efectiva va convergiendo en un punto. En este caso, el volumen de datos cae a un segundo plano, donde ya pasada esa convergencia, la cantidad de datos no aumenta el intervalo de tiempo para realizar dicha acción.

La escala de datos para la etapa A viene establecida para un conjunto de miles de bytes de datos, para la etapa B el conjunto a tratar es de millones de bytes y la etapa C concierne a centenas de millones de bytes.

4 Especificaciones y análisis

Una vez explicados los conceptos principales, se plantea los requerimientos previos a la realización del diseño. Se ha de recalcar que los objetivos impuestos son: la transmisión de ciertos datos para estimular un producto y la recepción de una posible respuesta.

4.1 Especificaciones

Las especificaciones son las siguientes:

- "Para la etapa de transmisión, se utilizan modelos de software para la generación de escenarios con una frecuencia de muestreo de 9.984 MHz".
- "El diseño utiliza 8 DACs para la transmisión de datos y 8 ADCs para recepción, todos los dispositivos funcionan con señales complejas. La resolución de las muestras es de 32 bits".
- "El diseño debe alcanzar al menos 3 GHz de ancho de banda tanto en transmisión como en recepción. Ambas etapas con la misma tasa de muestreo".
- "El diseño debe realizar ambas acciones (transmitir y recibir) en coherencia, es decir, al unísono entre señales". Con relación a esta especificación, ya que en el mundo físico no puede existir el desfase cero entre canales, la idea principal es conseguir que la diferencia de fase sea ínfima, alrededor de los 50 picosegundos (algo asequible para la calibración en hardware entre placa y producto).
- "El control del diseño se debe realizar a partir de un host externo a elección del diseñador".

Conocidas las especificaciones, el diseño se divide en dos procedimientos: Transmisión y Recepción. Para operar con ellas se trabajan de forma aislada. Sin embargo, como nexo de todo, se encuentra el módulo Zynq Ultrascale+ RF Data Converter para tratar el aspecto de radiofrecuencia.

Para facilitar la lectura, se ha decidido realizar la abreviatura del nombre del módulo Zynq Ultrascale+ RF Data Converter a "RFDC", AXI Direct Memory Access a "DMA", Transmisión a "Tx", Recepción a "Rx", protocolo AXI4-Stream a "AXIS" y protocolo AXI4-Lite a "AXIL".

4.2 Análisis

A partir de lo anterior, se pueden sacar las siguientes conclusiones:

- Se debe conseguir un ancho de banda mínimo a 3GHz tanto para transmisión como recepción. Teniendo en cuenta que en la etapa de transmisión, un requerimiento es utilizar como frecuencia de muestreo 9.984 MHz, la tasa de muestreo alcanzada tiene que ser múltiplo de esta frecuencia. Las etapas de procesado de la señal (interpolación y diezmado) varían esta tasa sobre un factor "x".

- La tasa de muestreo escogida viene determinada por su generación de frecuencia a partir de sintetizadores (en este caso LMX). Como se indica en el anexo A, es necesario que dicha tasa sea múltiplo de la frecuencia generada por la PFD (*phased frequency detector*), siendo esta frecuencia conocida como *Fpd*. Si no fuera así, no sería posible alcanzar los objetivos de coherencia por los problemas acontecidos en la generación del reloj.

Conociendo que dicha *Fpd* en la placa ZCU111 es 122.88 MHz, la tasa de muestreo elegida para el diseño es:

$$f_{TM} = 122.88\ MHz \times 320 = 3.19488\ GHz = \frac{32bits}{sample} \times 3.19488\ GHz = 3.19488\ GSPS$$

Ecuación 3: Cálculo de la tasa de muestreo

Para comprobar que los posibles problemas comentados anteriormente no interfieren, se valida a continuación la divisibilidad de los requerimientos. El hecho que sean múltiplos favorece en la generación de relojes la ausencia de espurios.

$$M1 = \frac{3.19488\ GHz}{122.88\ MHz} = 26$$

$$M2 = \frac{3.19488\ GHz}{9.984\ MHz} = 320$$

Ecuación 4: Cálculo de múltiplos para asegurar una tasa de muestreo adecuada

- Conociendo que en ambas etapas de procesado (interpolación/diezmado) la señal resultante trabaja en la misma tasa de muestreo, el factor a procesar es 320.
  Teniendo en cuenta que el RFDC, de forma interna (a través del DUC o en su defecto el DDC) contiene una etapa de interpolación/diezmado máximo de grado 8, es necesario realizar este proceso de forma extra fuera del RFDC. Para ello, es necesario utilizar módulos FIR Compiler (*cores* suministrados del catálogo de Vivado).

  Considerando que el proceso tanto en la interpolación como en el diezmado es el mismo, obtenemos que para ambos procesos es necesario una etapa extra de factor 40.

$$Interpolación/Diezmado\ Total = X_{RFDC} \times X_{Extra} = 8 \times 40 = 320$$

Ecuación 5: Factor total de Interpolación/Diezmado del diseño

  Esta operación se realiza en "cascada" ya que el desarrollo se sitúa en dos módulos aislados.

- Para la obtención de esta "coherencia entre canales" (también denominado como *MTS "multi-tile synchronization"*) se opta por implementar la funcionalidad a disposición obtenida en la documentación del RFDC [29].

- Para agilizar la configuración del diseño, el software utilizado es PYNQ, cuya ventaja principal es la facilitación de la implementación del diseño a partir de unas pocas líneas de código en lenguaje *python*. Esto es una de las principales razones por la que considerar a la ZCU111 como la placa ideal en este diseño, siendo una de las pocas tarjetas de evaluación con la compatibilidad de este *framework*.

5 Integración del hardware

Una vez presentados tanto los aspectos que se van a estudiar como los requerimientos, comienza la realización del diseño. Dependiendo de la función que se debe realizar (transmisión o recepción), su estructura es distinta.

La arquitectura del sistema se observa en la Figura 5-1.

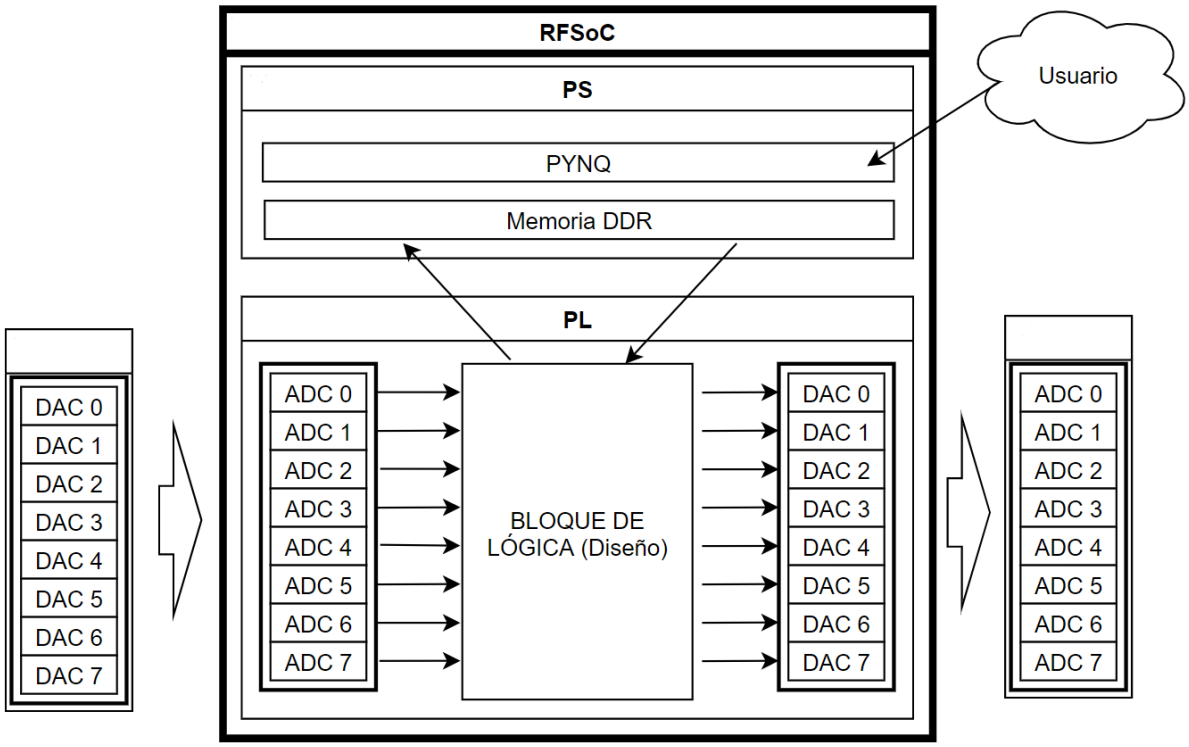

Figura 5-1: Arquitectura del diseño

El proceso de transmisión comienza por parte del usuario de introducir cierto escenario para la estimulación del producto. Este escenario es enviado a la memoria DDR (mediante PYNQ), a través del buffer de la DMA. Este módulo, en modo lectura, realiza la comunicación de PS a PL. Una vez llegado el volumen de datos al PL, este se procesa (teniendo en cuenta los requerimientos) y se expulsa a través de los DACs de la ZCU111 al producto en sí.

La recepción, en cambio, realiza el sentido inverso: recibe un volumen de datos de respuesta mediante los ADCs de la ZCU111, se procesan y se manda al PS (a través de la DMA en modo escritura). El usuario a través de PYNQ puede recibir los datos almacenados de la DMA (mediante su *buffer*) y procesarlo realizando la funcionalidad buscada.

Se entiende con ambos casos anteriores que tanto DACs como ADCs viven bajo el manto del módulo RFDC.

5.1 Objetivos del Hardware

Los objetivos que tiene el diseño son los siguientes:

- Distribución del volumen de datos (sin pérdidas) como equivalente a la señal o escenario tanto en transmisión como en recepción.
- Procesamiento de los datos para obtener la tasa de muestreo buscada, en este caso 3.19488 GSPS. Se aplica para ambas operaciones (interpolación y diezmado).

5.2 Estructura del PL

La arquitectura del diseño, como se ha comentado a lo largo de este documento, se han utilizado principalmente *IP Cores* suministrados por Xilinx. De esta manera, se reduce la carga de trabajo eliminando los tiempos para la descripción de los diferentes componentes. Obviando esto último, otra ventaja que añade es que son módulos verificados y de fácil adaptación por su aspecto parametrizable, en conclusión, son componentes "*plug and play*".

A continuación, se va a detallar brevemente las principales jerarquías que completan el cuerpo del diseño, para después explicar su funcionalidad y configuración a partir de su implementación en Vivado.

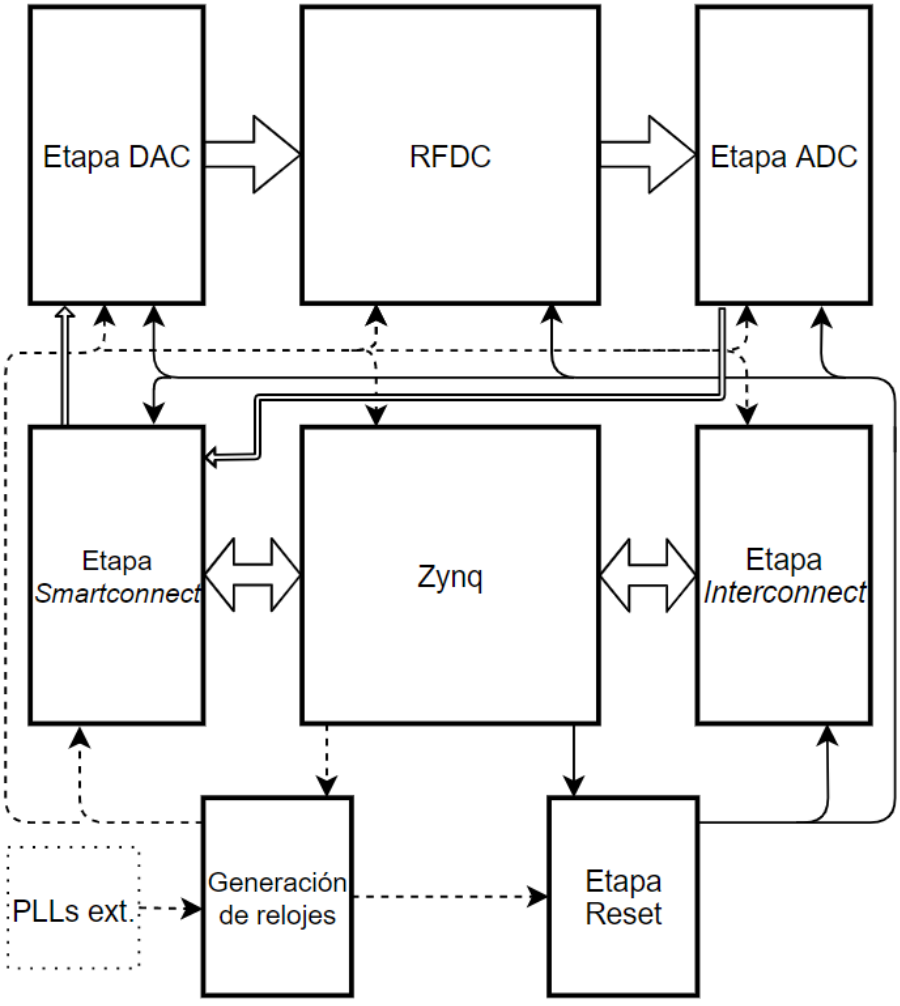

Figura 5-2: Estructura del PL

- **Zynq MPSoC**: Es el núcleo del diseño. Esta IP representa el apartado del PS, permitiendo su configuración y la de sus periféricos. Dichos periféricos incluyen interfaces AXI4 (“maestro” y “esclavo”) para manejar los distintos módulos del diseño.

- **Etapa “Smartconnect” e “Interconnect”:** Agrupa todos los módulos AXI SmartConnect y AXI Interconnect del diseño.

- **Etapa de generación de relojes:** Este bloque contiene todo lo referido al conjunto de relojes. Es aquí donde se contemplan los relojes pertenecientes a sintetizadores externos (LMK y LMX). Se ha de comentar que aquellos relojes generados en la Zynq se han trasladado también a este bloque, facilitando su localización a la hora de diseñar.

- **Etapa Reset:** Este bloque contiene todos los reset del diseño. Cada reloj generado en el PL tiene su reset equivalente.

- **Etapa DAC y ADC:** Estos bloques contienen todas las vías de procesado de los datos provenientes de los DACs y ADCs. Cada bloque trae integrado 8 caminos referidos al número de componentes para transmisión y recepción.

- **RF Data Converter (RFDC):** Es el bloque encargado de manejar la cadena de Radiofrecuencia. Se encarga de interactuar tanto para los DACs como los ADCs.

5.3 Implementación en Vivado

Una vez creado el proyecto, es necesario incluir los bloques que no vienen en el catálogo de *IP Core* de Vivado. Para ello, es necesario acceder al menú “*Settings*” en el “*Project Manager*”, apartado “*IP→Repository*” y añadir los directorios donde se encuentran estos módulos externos.

A continuación, comienza la descripción de la configuración establecida en los módulos. Dichas modificaciones son introducidas a partir de la API (*Application Programming Interfaces*) proporcionada una vez seleccionado cada bloque.

Antes de comentar las diferentes partes de la arquitectura del diseño, se ha de comentar que aquellas configuraciones detalladas en este documento son las que se han introducido a nivel de usuario. Aquellas no mencionadas se han dejado “por defecto”.

También para facilitar el entendimiento del lector, dependiendo de la densidad que integra cada etapa, se ha subdividido en diferentes partes para disminuir su complejidad. Además de aportar mayor claridad en las imágenes descartando la señal “reset” de cada módulo que viene en relación al dominio de frecuencia al que está operando (AXIS o AXIL).

5.3.1 Zynq MPSoC

La Zynq UltraScale+ MPSoC Processing System es el núcleo de cualquier sistema de procesamiento *Zynq UltraScale+*, su principal función es realizar la comunicación lógica entre el PS y el PL, además de dar soporte a la hora de integrar núcleos personalizados (para

ello el uso de *IP Integrator*). También conecta toda la señalización de interfaces con el resto del sistema.
En la Figura 5-3 se muestra la configurabilidad introducida en este bloque, siendo los módulos configurables los que tienen un color verde y los implementados con un *tick*. A partir de su interacción, se puede acceder a la configuración que contiene.

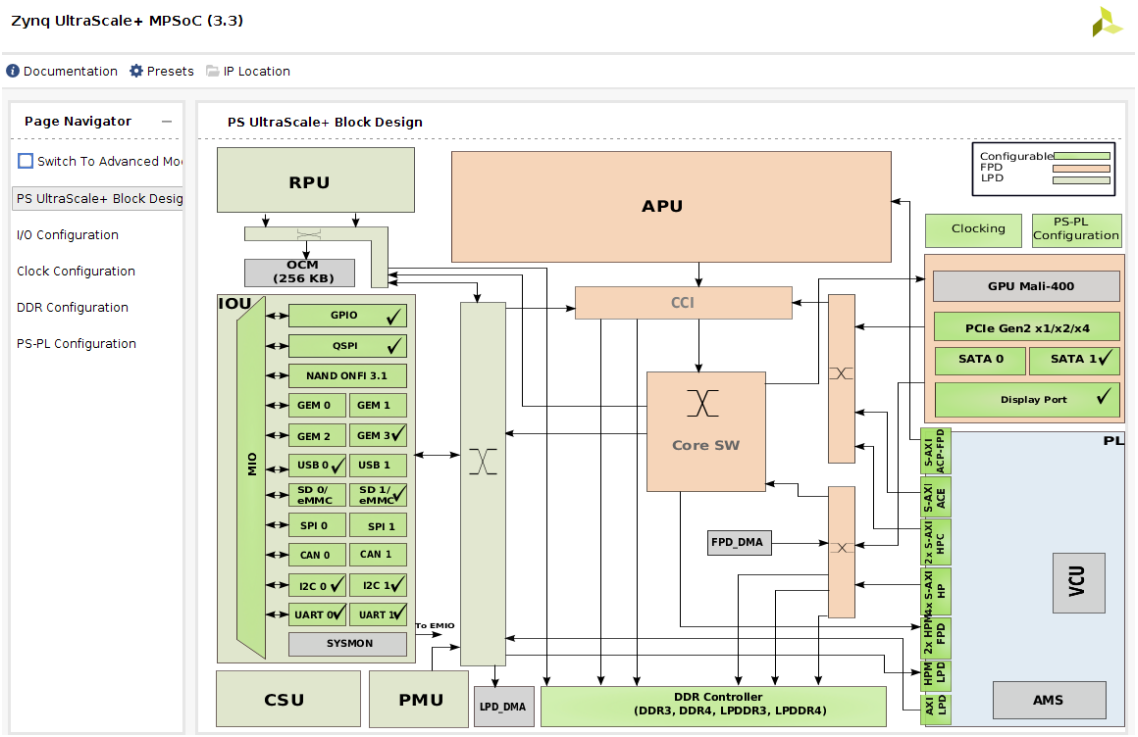

Figura 5-3: Diseño de bloques del módulo *Zynq UltraScale+ MPSoC PS*

- En el apartado "*I/O Configuration*" se pueden habilitar o deshabilitar los periféricos disponibles. Gran parte de esta configuración viene establecida por Vivado. Una de las ventajas que tiene este entorno es que es una interfaz de fácil interacción, tan solo con determinar qué placa de evaluación se está utilizando en el proyecto, él se encarga de personalizar los módulos a las características de la placa.
  Para el caso de realizar diseños de bajo nivel de personalización, se puede ahorrar tiempo a través de su opción de asistencia (*Designer Assistance available*) permitiendo también a los más inexpertos controlar la herramienta de una manera más intuitiva.

- Para el apartado "*Clock Configuration*" viene todo lo referido a la generación de relojes. En este caso, la configuración a tener en cuenta es la derivada a los relojes de salida (*Output Clocks*). Respecto al reloj del procesador y de los periféricos se han dejado por defecto.
  Lo más importante en este apartado son los relojes generados en el PL, es aquí donde se generan los relojes de los protocolos AXIS (para la etapa de Tx) y del protocolo AXIL. Estas frecuencias son 250 y 75 MHz respectivamente. El valor de la primera es debido a la necesidad en la etapa de Tx de enviar los datos a la etapa de procesado a una mayor frecuencia a la que lo procesa el RFDC (bus de datos AXIS del apartado DACs). Aunque es posible alcanzar una mayor frecuencia, se tiene constancia que pueden aparecer problemas en los módulos de almacenamiento (FIFO) cuando se trabaja con frecuencias en torno a 280 MHz. El valor de la segunda es debido a la

condición de que la frecuencia del protocolo AXIL debe ser menor o igual a cualquier frecuencia para el protocolo AXIS de todo el diseño.
Un punto importante es que dependiendo del PLL que esté seleccionado (IOPLL, RPLL o DLL), la frecuencia real que circula por el PL es la que muestra en la sección "*Actual Frequency*". Por tanto, para frecuencias escogidas que no son múltiplos a este PLL de referencia, la frecuencia resultante puede no ser tan exacta a la esperada.

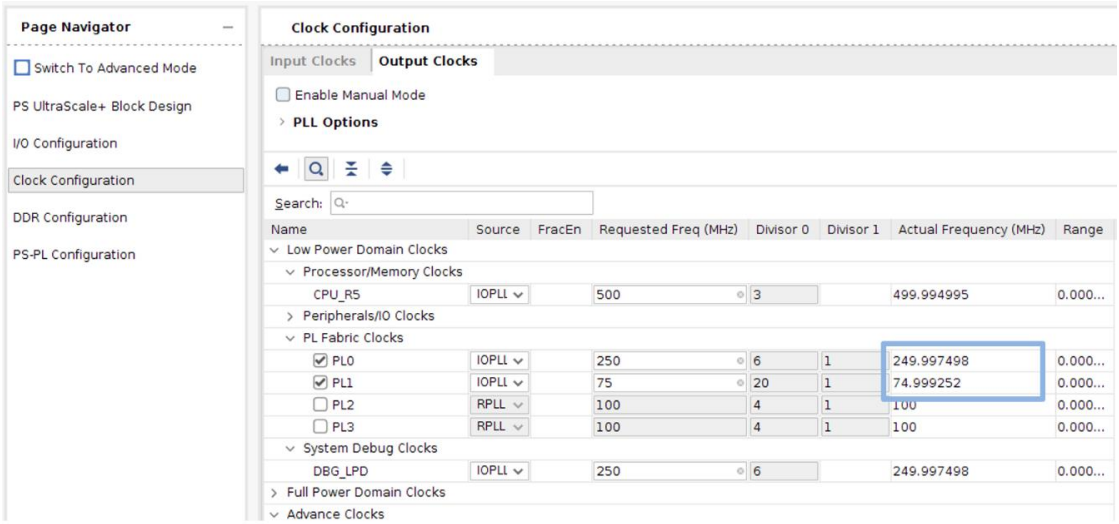

Figura 5-4: "*Clock Configuration" de la Zynq (PS)*

- El siguiente apartado es la configuración de la DDR ("*DDR Configuration*"), la cual se ha dejado por defecto.

- Por último, está el apartado de "*PS-PL Configuration*". Se accede a la sección "*General*" para habilitar la opción "*Interrupts*" (IRQ0). Esto es debido a que las DMA son módulos que trabajan con la funcionalidad de crear interrupciones en el procesador, de tal manera que pueden detener su funcionamiento para únicamente realizar la función que contiene en su ISR (*Interrupt Service Routines)*. En este diseño, es obligatorio su uso para poder trabajar (se conectan al PS a través del módulo AXI Interrupt Controller). Respecto al resto de las interrupciones se dejan por defecto.
  En la sección "*Fabric Reset Enable*", es necesario al menos tener activado uno para realizar la función de "*power-on-reset*".
  La última sección por modificar está relacionada con la conexión de interfaces "maestro" y "esclavo". Las interfaces "maestro" trabajan sobre el nivel del protocolo AXIL, se tienen conectadas 2 interfaces *HPM0* y *HPM1* (es necesario 2 por el número de componentes del diseño). Respecto al apartado de interfaces "esclavo", se debe de tener activado el máximo número. De esta manera, el número de transiciones en memoria se realizan por más "canales", obteniendo un mayor *throughput* en el sistema.

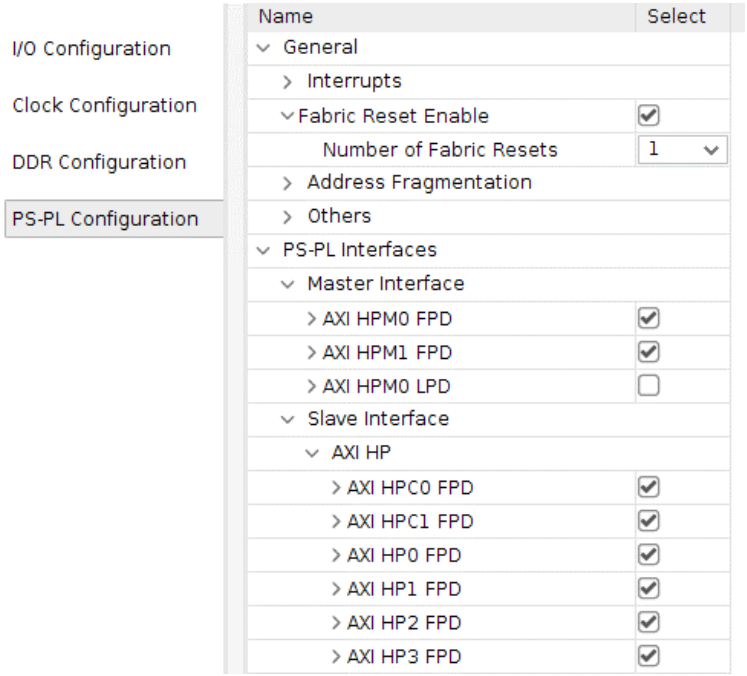

Figura 5-5: "*PS-PL Configuration" de la Zynq (PS)*

5.3.2 Bloques de reset

Algo imprescindible en la electrónica es el "reseteo" (reinicio) del diseño para la eliminación de valores almacenados. De manera que cada inicio en la placa de evaluación conlleva una inicialización de todos los bloques integrados que lo necesitan. Por ello, comparten esta estructura.

Cada bloque secuencial funciona a partir de una señal de reloj y, en general, con su reset respectivo.

Para conseguir esta obtención de reset se introduce el módulo Processor System Reset. Este bloque se encarga de proporcionar cada reset personalizado a cada frecuencia de reloj, funciona tanto para el procesador, la interconexión y los periféricos.

Este módulo trabaja de forma asíncrona al reset global ("*power-on-reset*"). Puede configurarse a nivel activo bajo o a nivel activo alto. En este caso, por la preferencia estándar que tiene la mayoría de los bloques del catálogo de Xilinx, funciona a nivel activo bajo.

En este diseño, hace falta 4 módulos con relación al número de relojes generados: *pl_clk_axis*, *pl_clk_axil*, *pl_clk_dac* y *pl_clk_adc*.

La configuración utilizada, es la que viene por defecto y se muestra en la Figura 5-6.

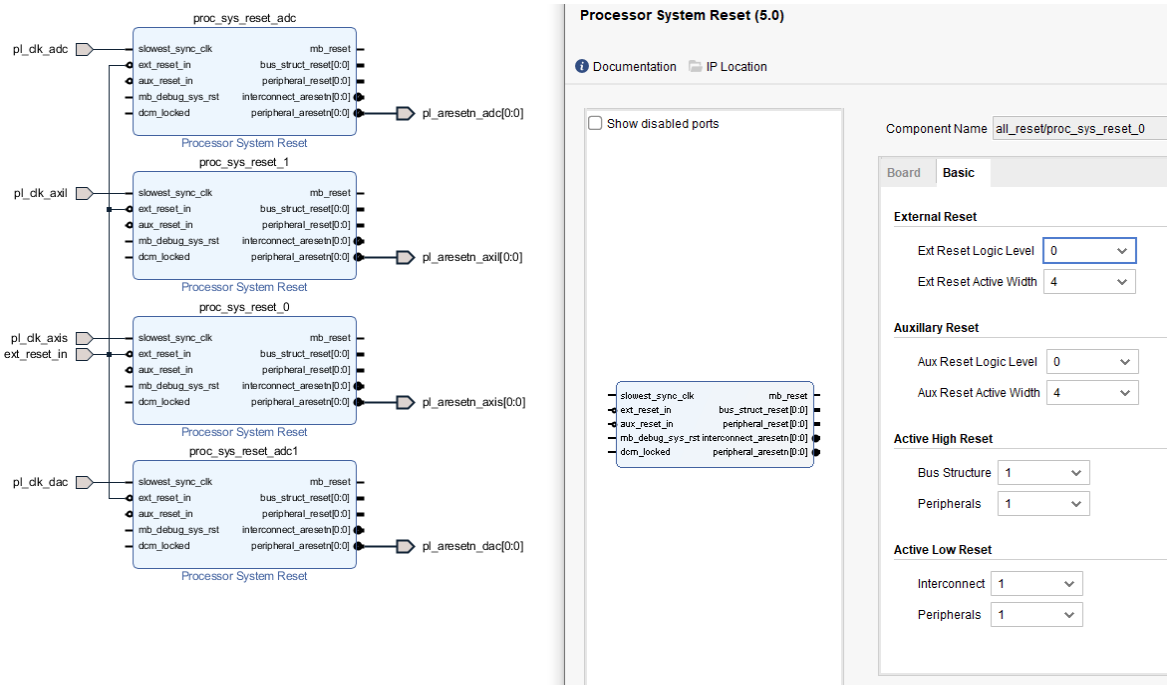

Figura 5-6: Configuración módulo *Processor System Reset*

De la Figura 5-6 se abstraen las conexiones principales:

- ***Slowest_sync_clk:*** Es el reloj al que se ha asignado el reset.
- ***Ext_reset_in***: Señal del reset global proveniente de la Zynq. Ejecutada con el reinicio de la placa.
- ***Peripheral_aresetn:*** Es el reset generado que va dirigido a los módulos destinados.

5.3.3 Bloques SmartConnect e Interconnect

Los módulos AXI Interconnect y AXI SmartConnect son los encargados de conectar la interfaz de propósito general a los diferentes periféricos en el PL. De tal manera que exista conexión de las diferentes interfaces "maestras" a un número de interfaces "esclavas" y viceversa.
La diferencia entre ellos es que AXI SmartConnect contiene una estructura más ligada al entorno Vivado, lo cual, optimiza mejor su funcionamiento.

A la hora de diseñar, si se trabaja en modo pasivo, es decir, dejando al entorno Vivado tomar el mando, tiende a escoger AXI Interconnect para la asignación de memoria para el ámbito del protocolo AXIL. De esta manera, reserva la utilización del AXI SmartConnect para un uso más avanzado. En nuestro caso, este segundo bloque se aprovecha para las transiciones en memoria de la DMA del protocolo AXIS.

Respecto a la jerarquía AXI Interconnect, conviven 2 bloques. Tienen un ancho de bus referido a la trasmisión de datos de 128 bits y conectan 16 y 17 interfaces "maestra" a su respectiva interfaz "esclava". Como se ha comentado anteriormente, trabajan a la frecuencia de *clock* del AXIL (75 MHz).

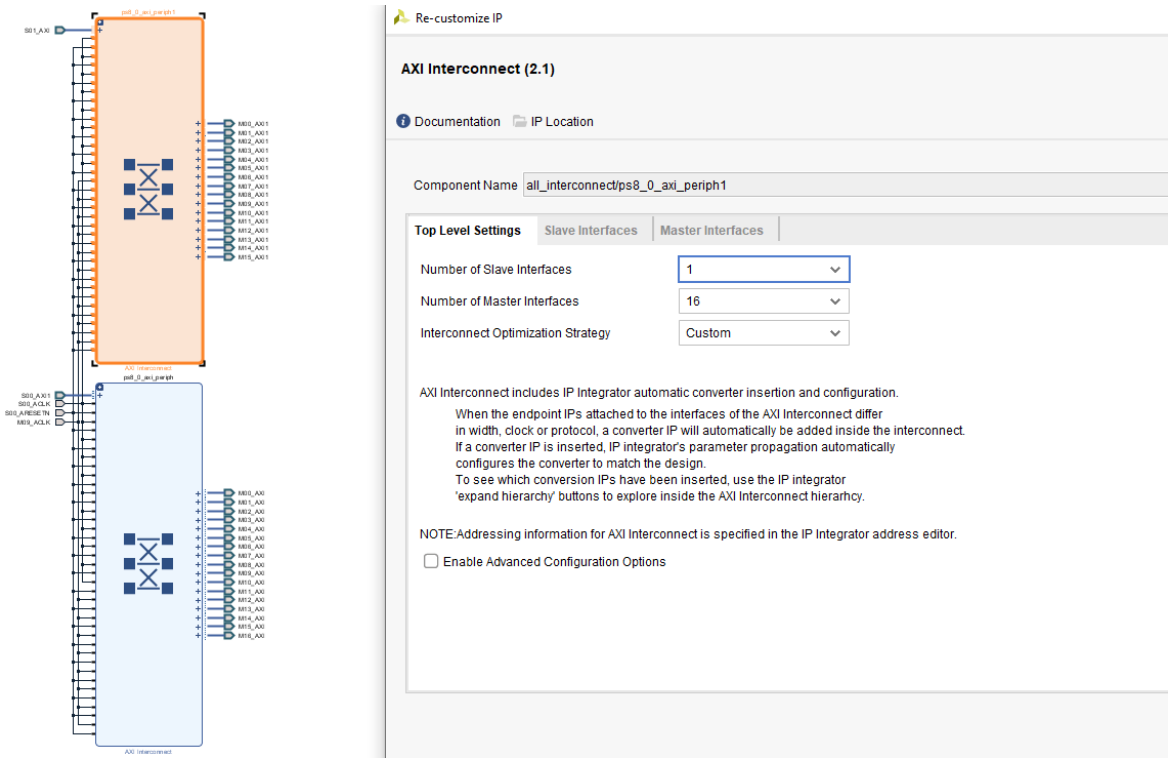

Figura 5-7: Configuración módulo Interconnect

La jerarquía del AXI SmartConnect contiene 6 módulos para alcanzar el máximo *throughput* disponible. El acceso en memoria depende del tiempo que tiene cada AXI SmartConnect en realizarlo. Este módulo fracciona el tiempo mediante el método "Round Robin", es decir, divide el tiempo del que dispone para realizar todas las transiciones por el número de componentes asignadas. A mayor número de asignación, mayor tiempo en el proceso. Por eso, la decisión de escoger el máximo número de interfaces PL-PS con su equivalente bloque AXI SmartConnect.

El ancho de bits depende de la operación realizada siendo para Tx de 128 bits y para Rx de 32 bits. La frecuencia a la que funcionan estos bloques viene determinada por el protocolo AXIS de la DMA implicada. En este caso son 250 MHz para Tx y 199.68 MHz para Rx.

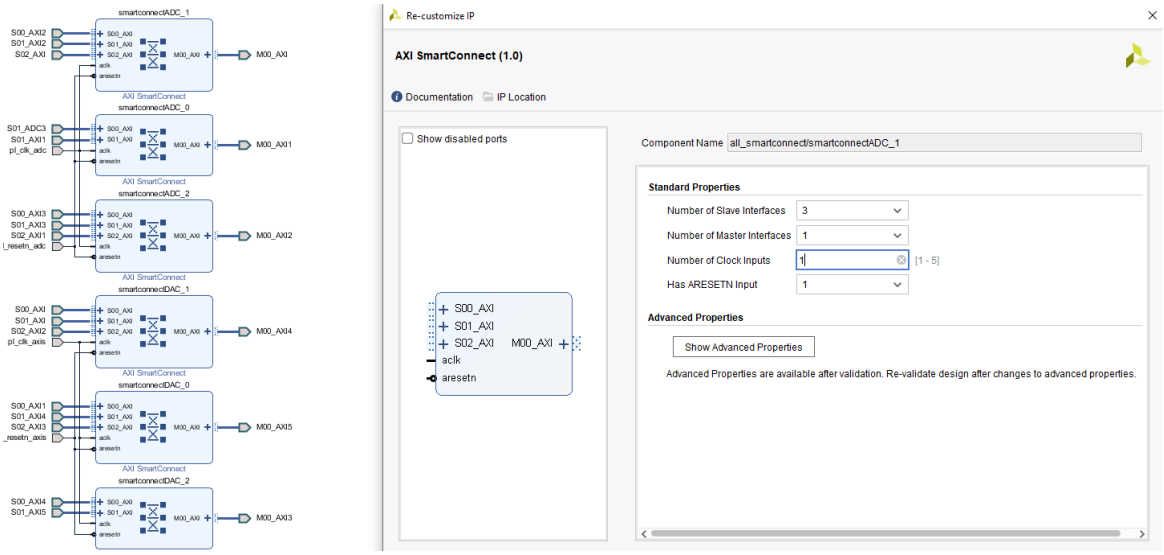

Figura 5-8: Configuración módulo SmartConnect

A la hora de enrutar las direcciones, hay que comprobar que todos los componentes del diseño han obtenido su acceso en memoria, y que además no han aparecido solapamientos.

Para visualizarlo, se utiliza el "*Address Editor*". En la Figura 5-9, se muestran los dos tipos de asignaciones. El recuadro rojo muestra el tipo de asignación derivado del bloque AXI SmartConnect y el recuadro azul muestra las asignaciones del AXI Interconnect.

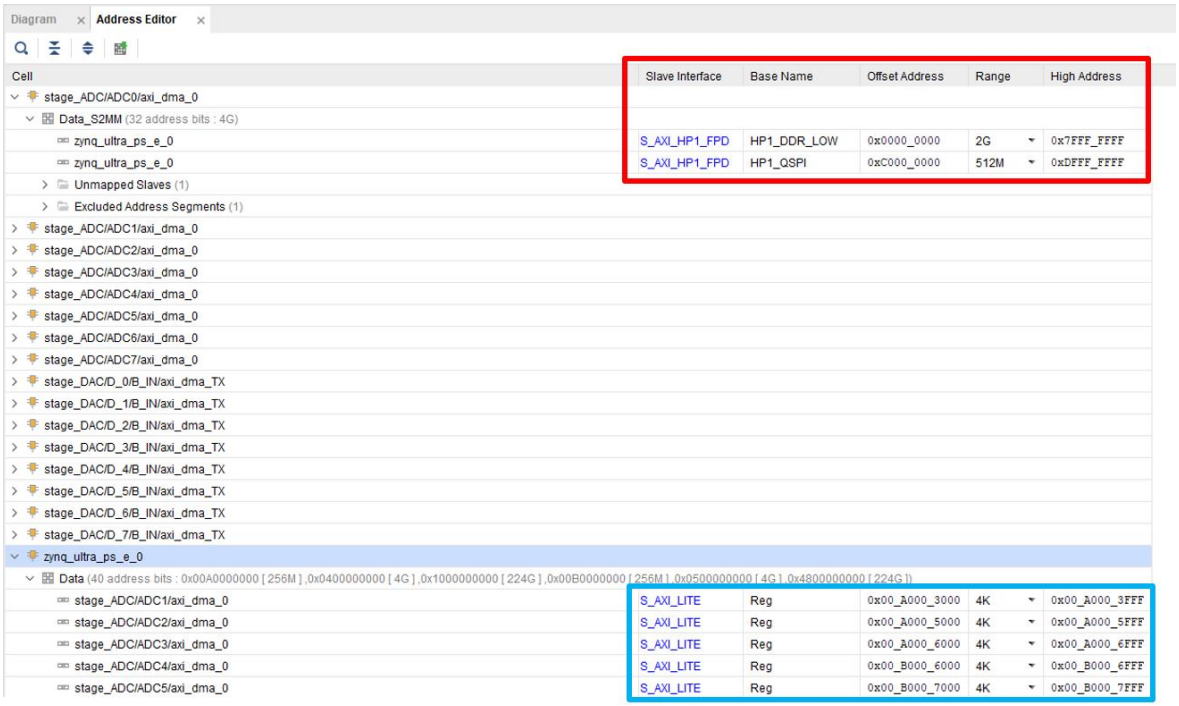

Figura 5-9: *Address Editor*

La manera en que los diferentes componentes se asignan en memoria recae en la fragmentación de direcciones. Dependiendo del espacio disponible y de los diferentes módulos que se compone el diseño, el procesador se encarga de traducir cada dirección y conocer las respectivas solicitudes de enrutamiento. La RFSoC contiene una RAM (DDR) de 4GB, fragmentada en dos secciones DDR_LOW y DDR_HIGH. Para este diseño, solo es necesario utilizar DDR_LOW para los accesos de la DMA. En el caso en que aparezcan problemas con alguna entrada de memoria, es posible resolverlo a partir de la exclusión o "desmapeo" de la entrada.

Para toda arquitectura en Zynq, la asignación de memoria funciona de forma idéntica. El término "*Offset Address*" indica el inicio del bloque de asignación de memoria y "*High Address*" indica el final.
Como se puede comprobar para el caso del recuadro rojo de la Figura 5-9, la DDR_LOW contiene un rango de 2GB, este comprende de:
[0x0:0x7FFFFFFF], es decir, 7FFFFFFF que es igual a un valor:

$$2^{31} = 2.14 * 10^9 = 2.14\ \text{GB} \approx 2\ \text{GB}$$

5.3.4 Generación de relojes

Este bloque contiene todo el grupo de relojes que circulan por el PL, tanto los generados internamente como externamente.

5.3.4.1 Relojes del PL

Los relojes del PL son generados por la Zynq (relojes dirigidos para el protocolo AXIS en Tx y protocolo AXIL de todo el diseño). La razón de trasladarlos aquí es para facilitar al diseñador la localización de todos los relojes, algo necesario cuando se trabaja con diseños de alta densidad de conexiones.
Uno de los principales defectos es que su alcance no es muy grande, su máximo valor es 333.33 MHz. Debido a esto, trae la necesidad de obtener otras fuentes con mayor alcance fuera del PL.

5.3.4.2 Relojes externos al PL

La cadena de Radiofrecuencia que trae incluida la ZCU111 engloba un conjunto de sintetizadores de máximo nivel (LMK04208 y LMX2594). Estos sintetizadores contienen en su estructura un PLL.

Un dispositivo PLL es un circuito constituido por divisores, donde tiene como función obtener una frecuencia en concreto. Para ello, realiza la autocorrección de una frecuencia inicial para alcanzar la frecuencia "objetivo". La forma para comprobarse es determinando que la diferencia de fase entre ellas es cero.

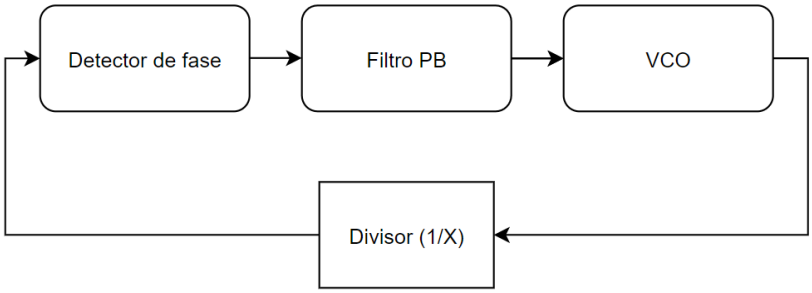

Figura 5-10: Estructura de un PLL

Los componentes externos que generan estas frecuencias de alto nivel son:

- El **LMK** (LMK04208) es el sintetizador de referencia con el que se obtienen tanto los relojes AXIS para el procesado de las muestras en el RFDC como los relojes generados por el LMX (siendo estos los encargados de suministrar al diseño la tasa de muestreo). El LMK tiene como frecuencia base 122.88 MHz y se encuentra en el diseño con el nombre de "PL_CLK_P_N".
  A partir del reloj "SYSREF_P_N" se obtiene la frecuencia común para conseguir la sincronización de todos los canales. Esta se administra como una señal periódica que interviene como evento común para que tanto *tiles* de DACs como ADCs operen en fase (coherentes). El cálculo de esta frecuencia se encuentra en el apartado 5.3.4.2.1.
  Tanto "PL_CLK_P_N" como "SYSREF_P_N" son puertos diferenciales y es necesario para obtener dicha coherencia que ambos se generen a partir de la misma

fuente.
A lo largo de este documento para abreviar los nombres utilizados, se va a referir al reloj “PL_CLK_P_N” como PL_CLK y “SYSREF_P_N” como SYSREF.

- Los **LMX** (LMX2594), son otro grupo de sintetizadores, tienen como función la generación de la tasa de muestreo y alcanzan valores cercanos a 15 GHz. En esta placa de evaluación existen 3.
  En nuestro diseño, tan solo es necesario trabajar con un solo LMX, ya que la tasa de muestreo para Tx como Rx es la misma.

Para configurar tanto el LMK como el LMX, se trabaja con el programa de software suministrado por *Texas Instruments* “TICS Pro”. En el anexo A se muestra la configuración.

En la Figura 5-11, se muestra la estructura de la ZCU111 para la generación de relojes.

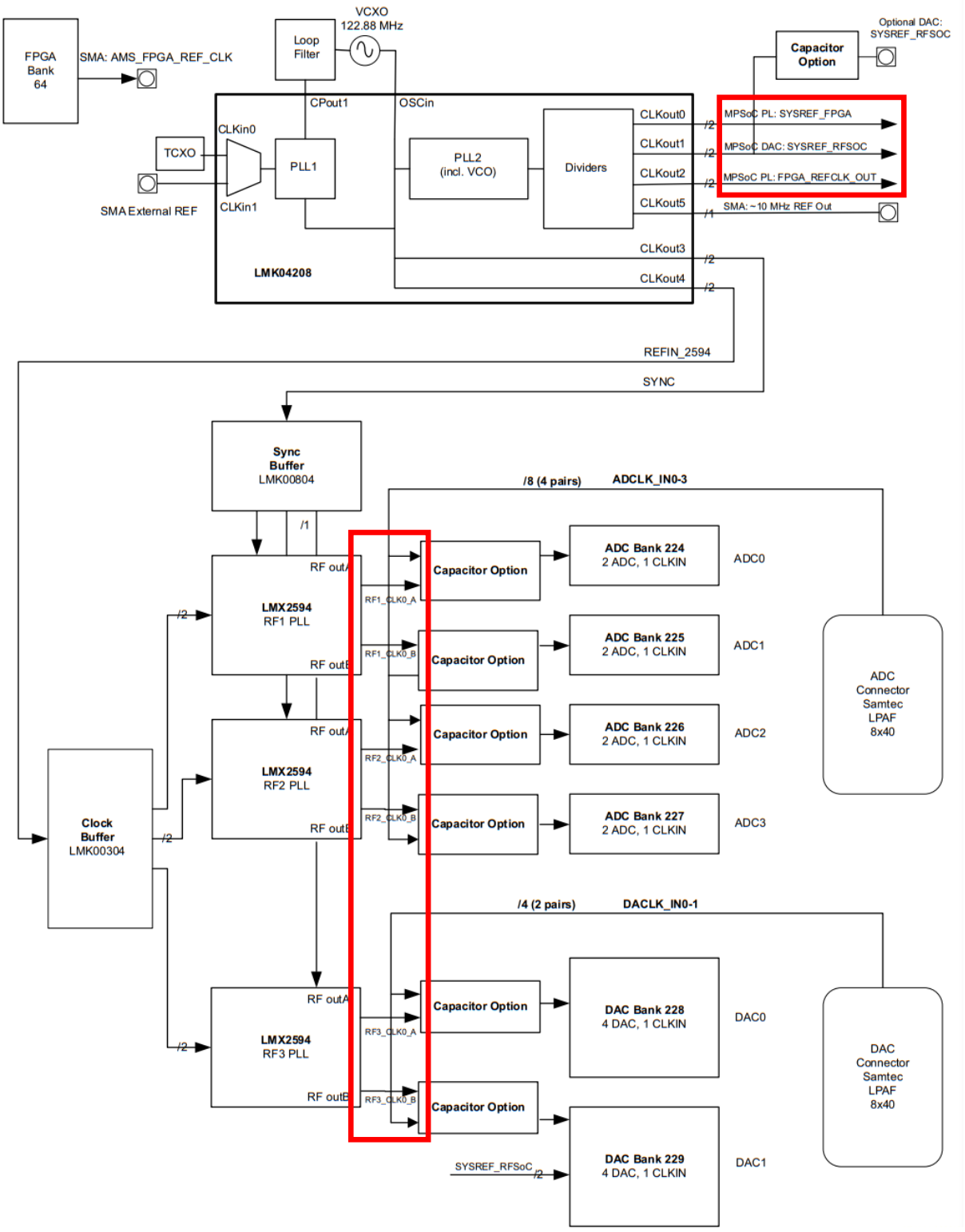

Figura 5-11: Estructura del reloj del RFDC (DAC y ADC) [15]

De esta figura se obtienen los relojes principales que afectan al diseño:

- ***SYSREF_FPGA***: es la señal SYSREF para conseguir la "coherencia entre canales".
- ***FPGA_REFCLK_OUT***: Es el reloj de referencia que trabaja a nivel AXIS para suministrar y recibir los datos del RFDC. Es la señal PL_CLK.
- ***RF_CLK***: Son las frecuencias generadas por el LMX para conseguir la tasa de muestreo.

5.3.4.2.1 Cálculo de la SYSREF

Para conocer el valor de frecuencia que se requiere, es necesario visualizar la documentación del módulo RFDC que nos suministra Xilinx [30].

Para su cálculo es necesario cumplir ciertos requisitos y son los siguientes:

- La señal generada debe de ser una onda cuadrada de alta calidad para facilitar que el reloj de muestra analógico lo capture de forma coherente.
- Respecto a su valor se debe de cumplir que:
  - Sea menor a 10 MHz
  - Tiene que ser múltiplo a la frecuencia resultante a la operación MCD(TM_{DAC} /16 , TM_{ADC}/ 16), siendo TM la tasa de muestreo elegida (3.19488 GSPS).
  - El valor de la SYSREF tiene que ser múltiplo a las frecuencias generadas en el PL para la interacción de datos con el RFDC, es decir, las frecuencias que provienen del reloj PL_CLK.

En el apéndice 9.1, se muestra un script sencillo para calcular este valor.

5.3.4.3 Estructura del bloque de generación de relojes

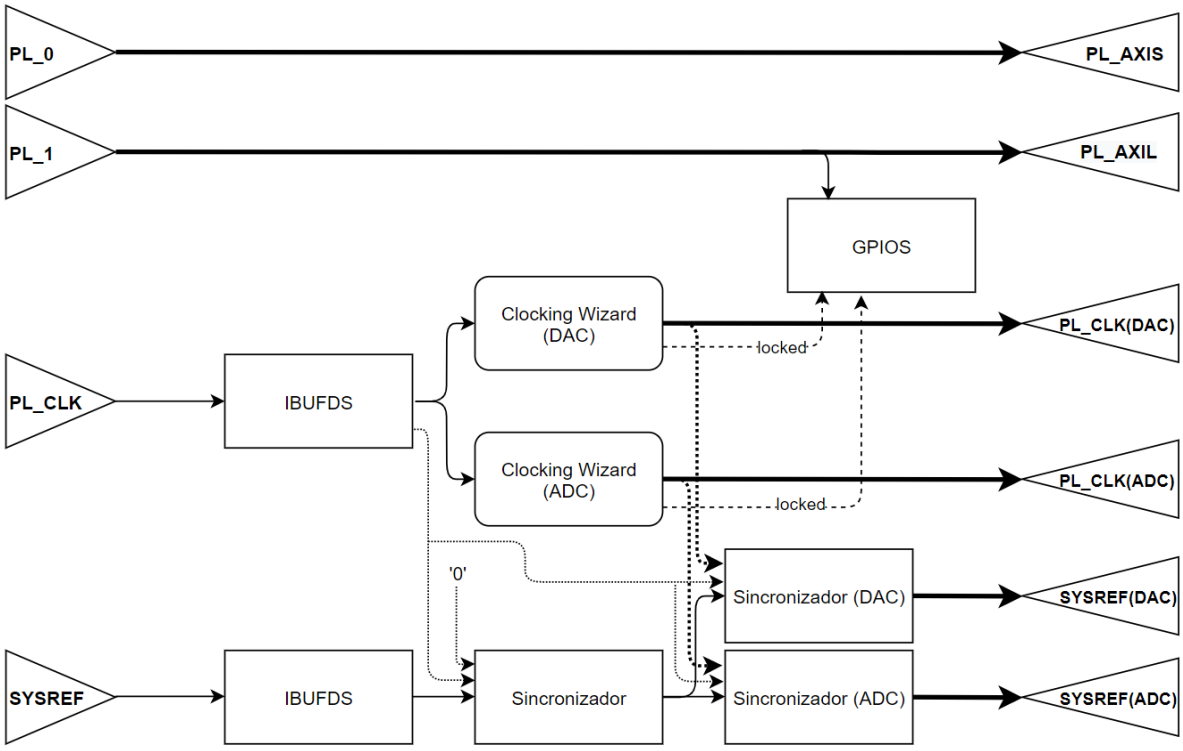

Figura 5-12: Estructura del bloque para la generación de relojes

Una vez realizado el proceso de enrutamiento por parte de los relojes generados fuera del PL (PL_CLK y SYSREF), es necesaria su conversión de relojes diferenciales (P y N) a modo común ("*single-ended*"). Para ello, se utiliza el módulo Utility Buffer con formato "IBUFDS".

Como se ha ido comentando a lo largo de este documento, es esencial que tanto PL_CLK como SYSREF se generen desde el mismo módulo para que guarden la misma fase. El proceso en las que se operan ambas señales es distinto.

El reloj PL_CLK es el encargado de suministrar al RFDC la frecuencia para procesar tanto los datos para los DACs como para los ADCs de la placa. Dependiendo de la configuración elegida en el RFDC, la frecuencia que necesita el AXIS del "maestro" (*m_axis*) varía, lo cual implica la utilización del módulo Clocking Wizard.

El Clocking Wizard es un módulo que permite al usuario operar con un PLL (o en su defecto, un MMCM) internamente en el diseño. Tiene la misma funcionalidad que un PLL común, es decir, genera una frecuencia a partir de la entrada de otra.

Su configuración es la siguiente:

- En la sección "*Clocking Options*" se ha elegido de componente generador el MMCM, ya que este obtiene mejores resultados en niveles de *jitter*. La siguiente opción es "*Frequency Synthesis"* para obtener frecuencias distintas al reloj de entrada. La opción "*Phase Alignment"* permite guardar la fase entre el reloj de entrada y el generado. Por último, se ha optado por configurar la opción "*Jitter Optimization"* como balanceado.

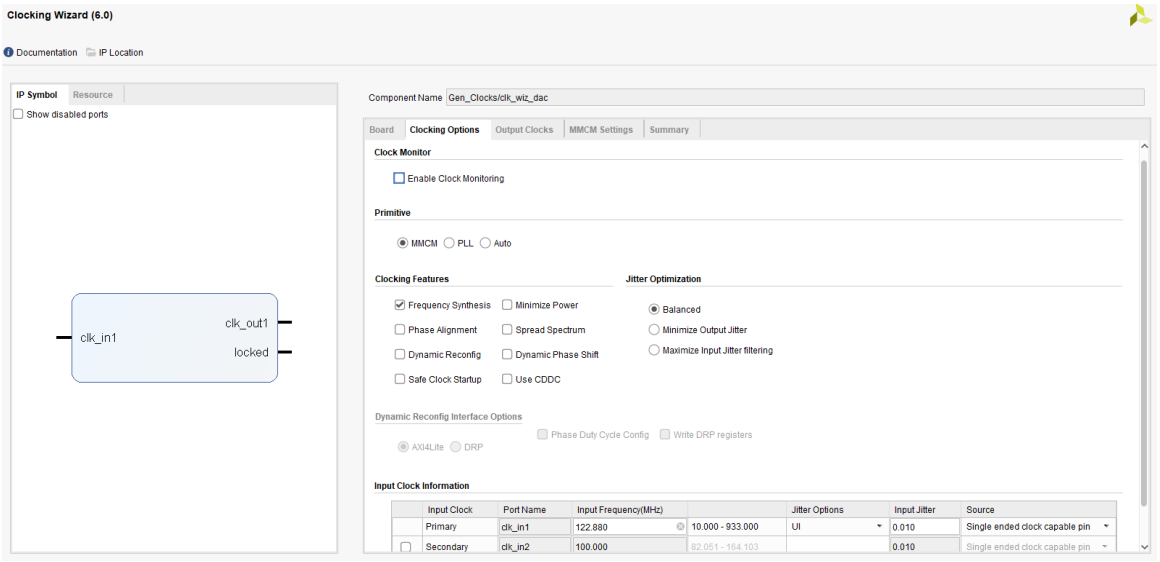

Figura 5-13: "*Clocking Options*" del módulo Clocking Wizard

- En la sección de "*Output Clocks"* se escoge la frecuencia generada. También se ha habilitado la opción "*locked*": una señal de salida que muestra si ha obtenido su cometido, mostrando "1" en caso afirmativo y "0" en caso contrario. En nuestro caso, realizamos estas lecturas como modo informativo mediante GPIOs.

Aunque este módulo puede generar hasta 7 relojes distintos, se ha optado por trabajar independientemente (199.68 MHz y 79.872 MHz). De esta manera, permite conocer el problema por aislado.

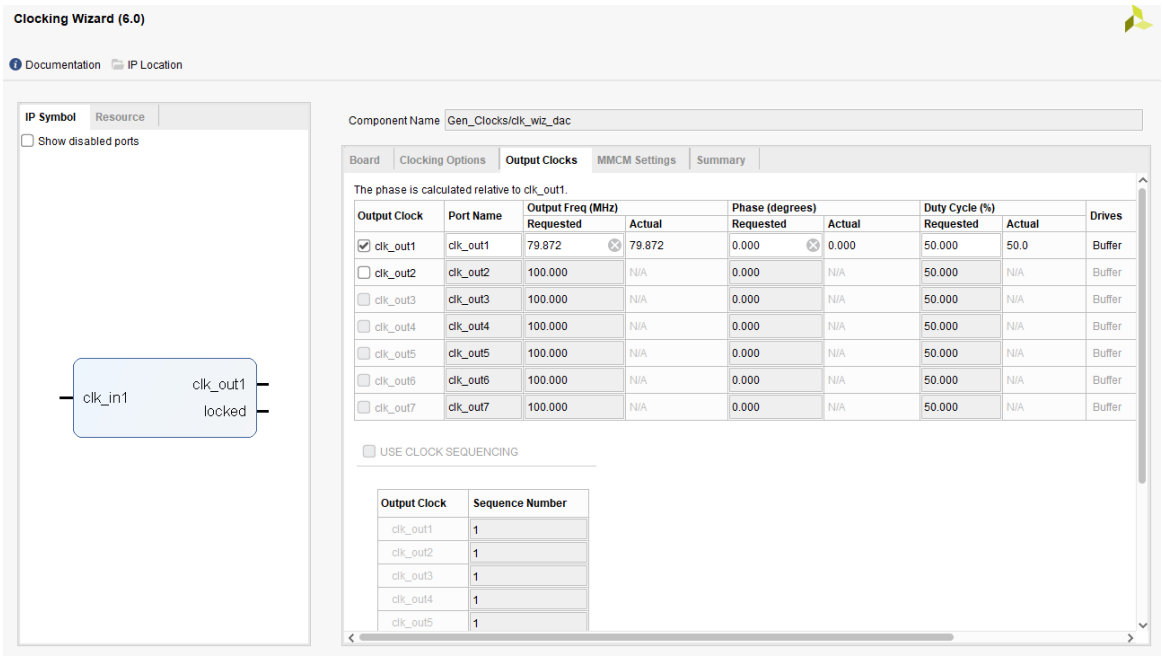

Figura 5-14: "Output Clocks" del módulo Clocking Wizard

Para acabar con esta generación de relojes queda pendiente el procesamiento de la SYSREF. La señal SYSREF se trata de un evento de sincronización entre *tiles*, es decir, es esencial que toda la cadena de RF comparta el mismo estado común. De tal manera, que internamente se actualicen los NCO de cada *tile* y que tanto DUCs como DDCs trabajen en sincronía para alcanzar el objetivo.

Para configurarlo, la guía del RFDC muestra dos situaciones:

- La primera situación muestra el proceso de sincronía que se debe realizar a la SYSREF, cuando el reloj AXIS del RFDC es el mismo que el generado por el LMK (PL_CLK). En este caso tan solo es necesario operar con un biestable.

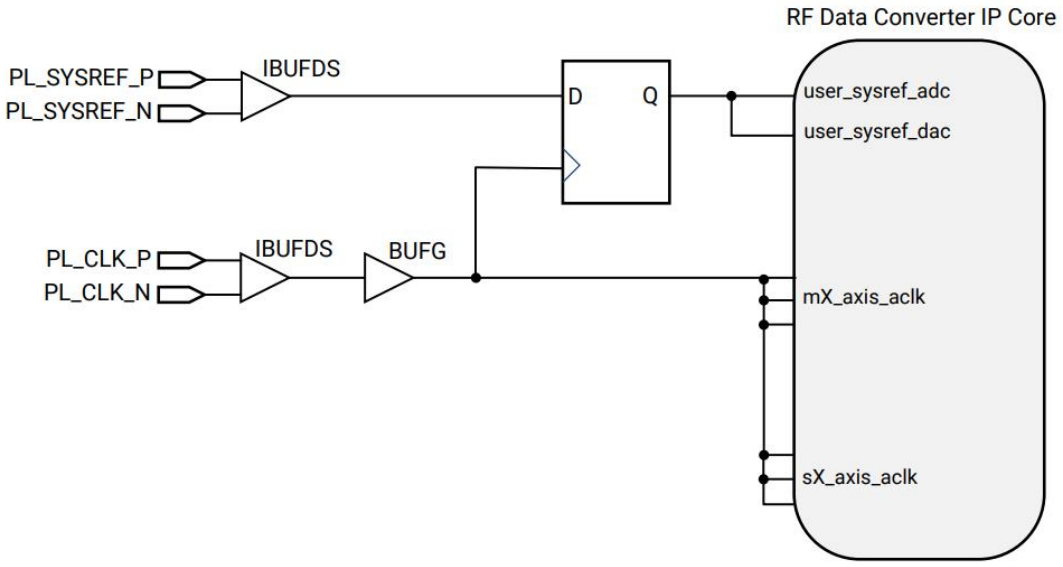

Figura 5-15: Primera situación de configuración del RFDC (SYSREF) [16]

- La segunda situación muestra el formato cuando se trabaja con dos frecuencias distintas al PL_CLK.

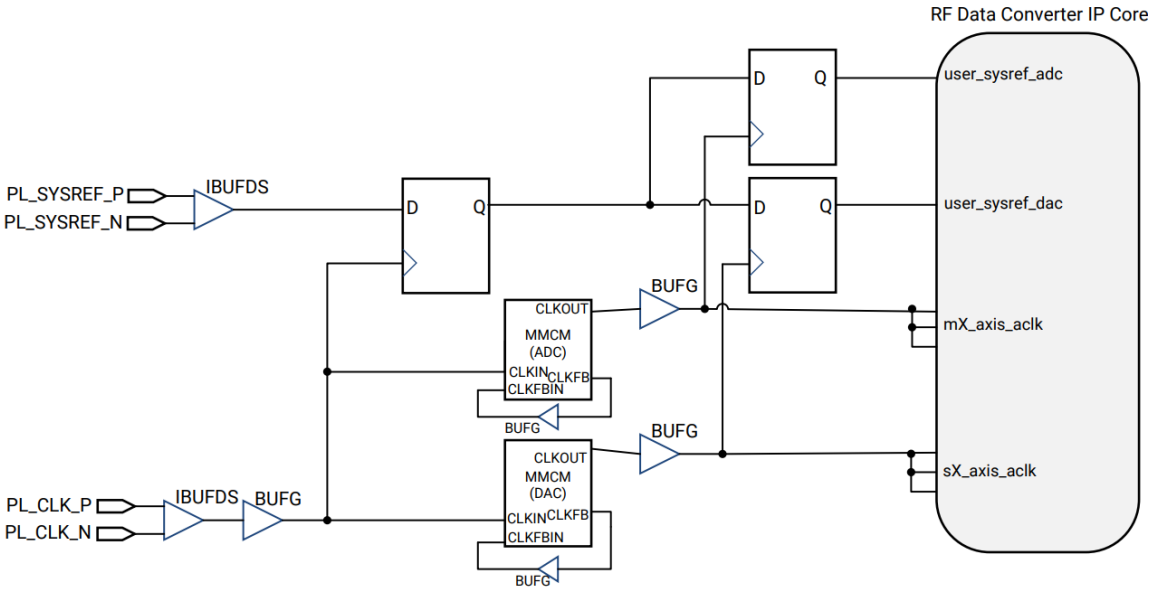

Figura 5-16: Segunda situación de configuración del RFDC (SYSREF) [17]

Sin embargo, este proceso que muestra la guía puede tener problemas cuando se trabaja con altas frecuencias, apareciendo el fenómeno de "metaestabilidad", es decir, el dato no llega a establecerse antes del siguiente pulso de reloj. Esto provoca que el valor introducido opere erróneamente o tenga un comportamiento inesperado (no es posible determinar si es "1" o "0").

Para evitarlo en nuestro diseño, se ha introducido un circuito de sincronización.

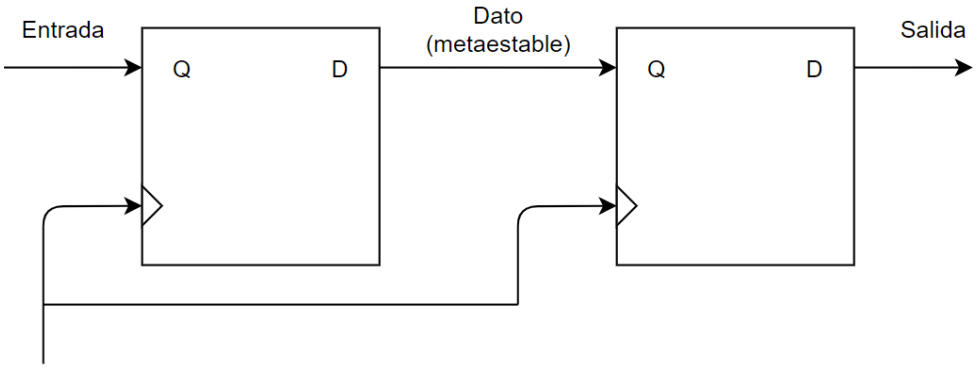

Figura 5-17: Circuito de sincronización

Este circuito sincronizador se trata de una serie de biestables para resolver el problema. En este diseño, dicho circuito está integrado en el módulo "sync", el cual, el usuario introduce por genérico el número de biestables que se van a situar en serie. Para asegurar que ninguna frecuencia pueda ocasionar algún tipo de problema, se han introducido 4 biestables.

Para acabar, se debe añadir que cualquier señal externa introducida en la placa es asíncrona a esta. Por tanto, es necesario que toda señal que provenga de fuera se sincronice al funcionamiento del PL.

5.3.5 Etapa de Transmisión (DAC) y Recepción (ADC)

Las etapas de Transmisión y Recepción trabajan el procesamiento de los datos. Dependiendo de la función a realizar, se establece la comunicación desde PS a PL (instrucción de lectura) o desde PL a PS (instrucción de escritura).
Estas comunicaciones en memoria provocan cierta latencia en el proceso (algo que se debe tener en cuenta a la hora de diseñar), produciendo un comportamiento no adecuado de cara al diseño. Estos retardos se reflejan de una escala de milisegundos hasta microsegundos. Sin embargo, trabajar con altas frecuencias ocasiona un problema severo en la comunicación. Estos problemas se visualizan claramente como brechas de datos (huecos en el flujo de datos), ya sea en la etapa de transmisión vaciándose el circuito o en la etapa de recepción mediante la detención del proceso cuando el circuito está a la espera de reanudar el envío de datos.

Para resolver este problema se trabaja con módulos de almacenamiento. Su objetivo principal es que siempre exista flujo de datos aun cuando existan asignaciones en memoria.

Este diseño se basa en trabajar con módulos que comparten el formato "*Blocking*", esto descarta la posibilidad de pérdida involuntaria de datos en cualquier proceso. La transmisión de datos es controlada por la interfaz "maestra" a la que se dirigen los datos, es decir, el último módulo con este formato comanda la serie de módulos antepuestos, estableciendo una jerarquía de disponibilidad que interrumpe el proceso si el siguiente en el proceso no está preparado.

Ya que todo el diseño se trata de un sistema coherente, las vías de transmisión para ambas etapas han de serlo también. Por ello, el proceso tanto para Tx como Rx es independiente y replicable para cada componente DAC/ADC. El número de vías de datos (*datapath*) que se dispone es el igual al número de componentes establecidos (8).

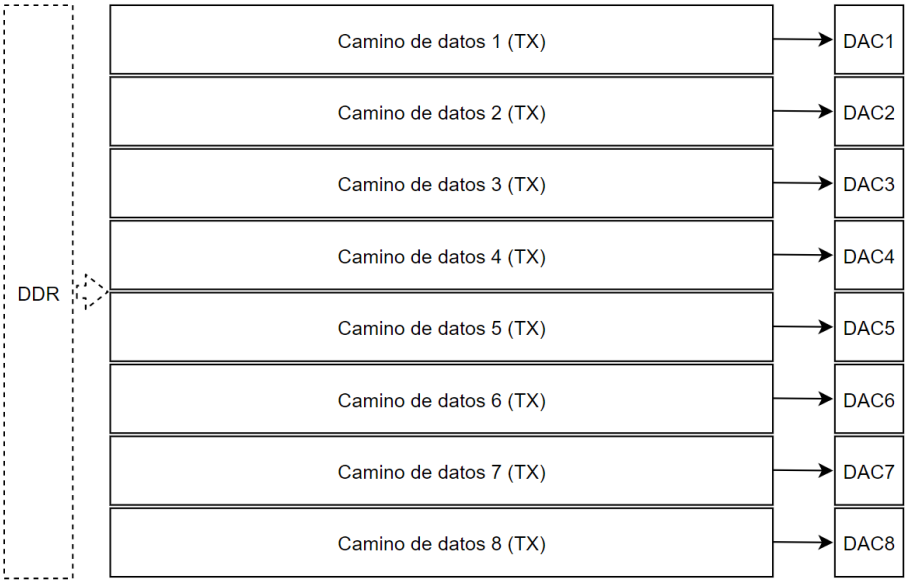

Figura 5-18: Estructura de la etapa de transmisión

Figura 5-19: Estructura de la etapa de recepción

5.3.5.1 Etapa de Transmisión

Esta etapa tiene como objetivo la estimulación de la interfaz de recepción del producto prefabricado. Para ello, es necesario la introducción de datos a partir de los DACs de la ZCU111 (siendo esto el punto final del proceso). Esta generación de datos se basa en introducir ciertos escenarios para validar el funcionamiento.

Se han implementado dos formas para la generación de datos. Para cada caso interviene un módulo determinado:

- El módulo DMA tiene como función transmitir el escenario que se ha preestablecido por software. Esto permite al diseñador trabajar cómodamente a través de un lenguaje de programación.
  El procedimiento principal es introducir el volumen de datos al hardware. Para ello, se trabaja con el buffer interno de la DMA, siendo este el vehículo de transmisión entre el apartado PL-PS. Algo importante a tener en cuenta es que las estructuras que trabajan la lógica binaria tienen muy poca flexibilidad. Por ello, cae a manos del diseñador realizar los debidos ajustes que realicen ese total entendimiento entre el diseñador y la FPGA. El principal ajuste que realizar es la conversión al formato digital complejo, es decir, 16 bits para parte real y otros 16 bits para parte imaginaria (en total 32 bits). Y también, su debida extensión de signo si es que lo hubiese.

- La otra forma de obtener esta generación de datos es incorporando el módulo DDS Compiler. Este módulo genera de forma continua una onda sinusoidal asemejando una señal. Como el requerimiento es obtener una tasa de muestreo a partir de una frecuencia de muestreo (9.984 MHz), dicho dato se le es introducido por configuración.
  El objetivo de este último módulo es facilitar al diseñador la comprobación de la sincronización entre canales (*multi-tile synchronization*). Ya que se trata de una única señal, permite analizar a los distintos canales su desfase medido, validando o no este requerimiento. Una vez realizado, comenzaría la transmisión de los diferentes escenarios para estimular el producto.

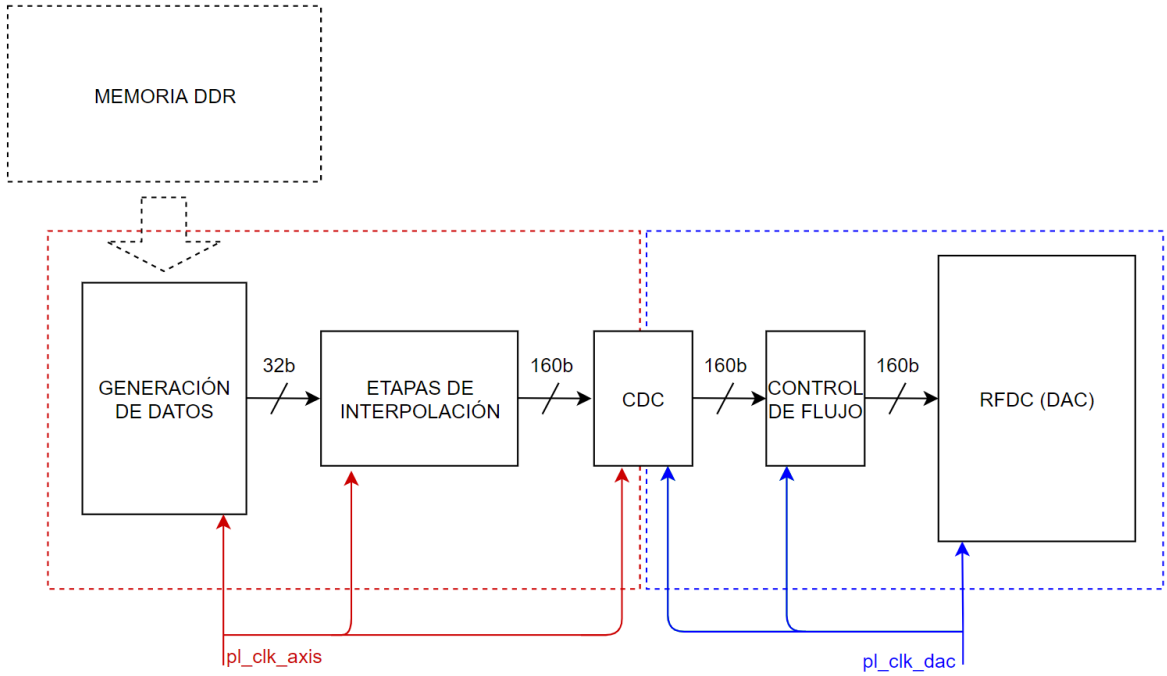

Figura 5-20: Diagrama de bloques para la etapa de transmisión

Esta estructura trabaja a partir de dos dominios de frecuencia: "alta frecuencia" y "baja frecuencia". El objetivo de esta transición es almacenar muestras en el proceso para solventar los problemas surgidos por el *delay* cuando se accede en memoria por parte de la DMA.

Las frecuencias que intervienen son:

- ***pl_clk_axis**:* Es la frecuencia a la que funciona el flujo de datos (*stream).* Esta frecuencia se ha generado en el PL (mediante el procesador Zynq). Su valor es 250 MHz.

- ***pl_clk_dac**:* Es la frecuencia a la que interactúan los buses de datos AXIS en el RFDC. Con esta frecuencia se procesan las muestras en la cadena de RF. Aunque la frecuencia base viene originada del LMK, el valor final de la frecuencia es obtenido a partir del PLL integrado en el PL (*Clocking Wizard*). Su valor es 79.872 MHz.

Cabe destacar que aquellos módulos que interactúan con el protocolo AXIL (***pl_clk_axil***), este sigue presente en todo el diseño, aunque no se muestre en las imágenes. Su frecuencia es 75 MHz.

5.3.5.1.1 Generación de datos

Como ya se ha comentado previamente, existen dos fuentes a la hora de alimentar el circuito: DMA y el generador de ondas sinusoidales (DDS). La forma de elegir qué componente es el que transmite se realiza a través de un *switch* (AXI4-Stream Switch).

La generación de datos necesita realizar su función a una frecuencia mayor a la que se le introducen los datos en el RFDC. La frecuencia estipulada en esta etapa es la frecuencia del AXIS (250 MHz). El hecho que se establezca esta velocidad de generación produce que los módulos antepuestos al RFDC trabajen a una mayor velocidad en comparación a la que lo hacen los DAC. Esto produce que se acumulen los datos en el proceso, para ello el uso de

módulos de almacenamiento (FIFO). Su objetivo es resolver los problemas que producen las brechas de datos suministrando en el circuito las muestras de su interior.

Esto conlleva la realización de análisis para determinar el intervalo de tiempo estimado del que necesitan las DMAs para completar las lecturas en memoria. Dependiendo de estos resultados, se incorpora mayor o menor almacenaje para garantizar el proceso.

Este problema anterior no sucede con la opción del módulo DDS Compiler, ya que genera datos de manera continua desde el apartado del PL.

La estructura se establece de la siguiente forma:

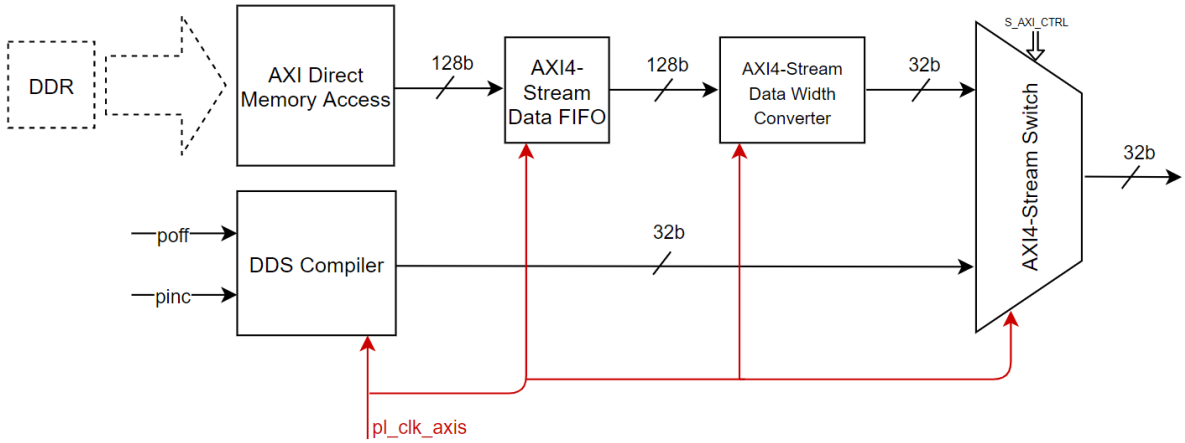

Figura 5-21: Estructura de la etapa "Generación de datos"

5.3.5.1.1.1 AXI Direct Memory Access (Modo Lectura)

Este módulo es el encargado de acceder a la memoria (DDR). Su principal función es copiar un bloque de memoria del apartado del PS al PL. La transmisión se realiza a través de la lectura en memoria del AXI4-Memory Mapped de la interfaz "maestra" (MM2S). Aunque este módulo puede trabajar en paralelo realizando múltiples transacciones o realizar la comunicación bidireccional, es decir, la comunicación PS-PL y PL-PS. Para facilitar el entendimiento del diseño, se ha querido mostrar de una manera más sencilla aislando cada proceso.

La configuración es la siguiente:

- Los módulos DMAs, además de ser los encargados de la transición de lectura o escritura, contienen una estructura también de almacenaje, la cual se irá vaciando a lo largo de la transmisión de datos. Este diseño se ha optado por la máxima capacidad límite que se proporciona (26 bits). El ancho de la dirección de la memoria se ha establecido por defecto (32 bits).

- Para establecer el formato de funcionamiento para comunicación del PS al PL, se habilita la opción de "*Enable Read Channel"*. La opción "*Memory Map Data Width*" indica el ancho del bus para el mapeo de memoria, aunque viene con diversas configuraciones, hay que tener en cuenta la estructura en la que se compone la ZCU111 (Figura 2-13), donde viene establecida por 128 bits. La opción "*Stream Data Width*" establece el ancho de bus del AXIS, es decir, la longitud de palabra a la

que se transmiten los datos a la salida de la DMA en el PL. Se ha escogido 128 bits. Sin embargo, viene totalmente relacionado por la siguiente opción "*Max burst Size*". Esta opción determina el número de ráfagas de la que se compone la transición, es decir, si se quiere alcanzar el mayor rendimiento de la tasa de transferencia efectiva (*throughput*) en el sistema, es necesario que se introduzca el mayor valor (256). La introducción de este valor limita el ancho de bus del AXIS a 128 bits (valor escogido).
La opción "*Allow Unaligned Transfer*", establece o no el funcionamiento del DRE (*Data Realigment Engine*). El DRE se encarga de realinear los datos de tal manera que la primera posición obtenida de la memoria del AXIS es el primer valor válido del conjunto de datos.

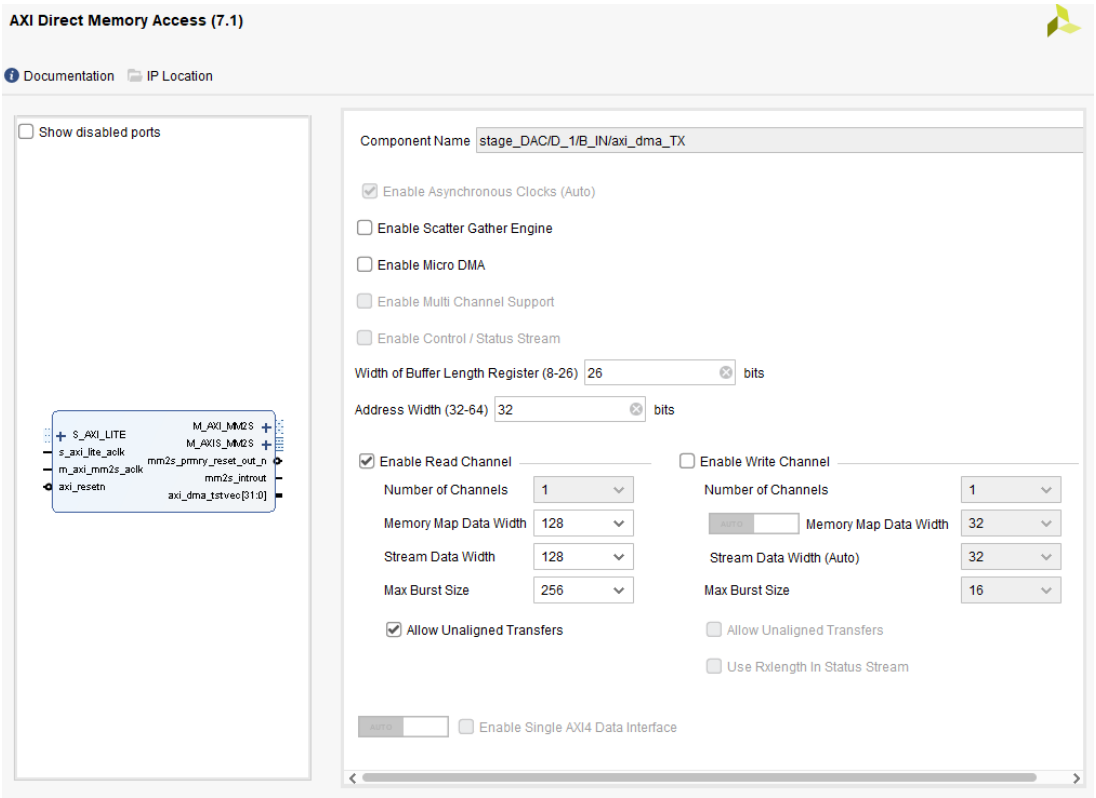

Figura 5-22: Configuración del módulo AXI Direct Memory Access (Modo Lectura)

5.3.5.1.1.2 DDS Compiler

El DDS Compiler es un módulo de generación de fase de alto rendimiento, el cual trabaja en el formato de interfaces AXIS. El objetivo principal es la generación de una onda como medida de comprobación para validar el correcto funcionamiento del diseño. Su funcionamiento es recrear dicha onda analógica utilizando parámetros digitales.

La estructura de este módulo se basa en conseguir una onda en concreto a través de un acumulador de fase, siendo este un contador de módulo M que tiene 2^M estados digitales. Este número de estados establece la división de puntos en los que divide toda la región angular (0:2π), estableciendo la resolución de fase.

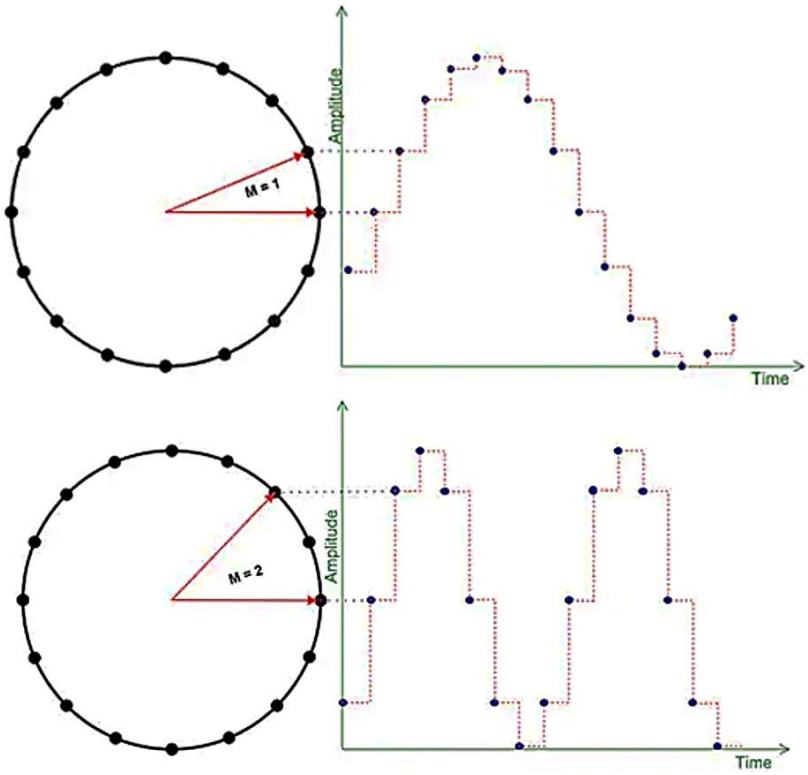

Figura 5-23: Ejemplo de generación de onda a partir del módulo DDS Compiler [18]

La configuración de este módulo tiene diversas opciones. En este caso, se ha introducido la opción "*Phase Generator and SIN COS LUT*" para obtener una señal sinusoidal. El ancho de las muestras (incluyendo I y Q) tiene de máximo 16 bits, lo cual, es necesario una extensión de signo para obtener el tamaño utilizado del diseño. Se ha establecido que su frecuencia de muestreo es 9.984 MHz (referente al modelo de software). También, se ha incluido que exista programabilidad a partir del bus AXIS para ser manejado desde software.

Para trabajar con un módulo más simplificado a las necesidades del diseño, se ha generado un nuevo *core* denominado "dds_gen". Este módulo se ha configurado para poder transmitir ondas de 32 bits (valor equivalente al ancho de una muestra). La forma de introducir los parámetros que modifican la forma de onda se ha configurado de forma más intuitiva. En este caso, se ha dividido el bus AXIS del módulo para obtener por aislado los dos parámetros principales, los cuales son:

- **Pinc**: Este valor establece la resolución de fase (*phase increment*) modificando en la onda la frecuencia a la que oscila.
- **Poff**: El *phase offset* es la opción que añade cierto *offset* en la generación de onda.

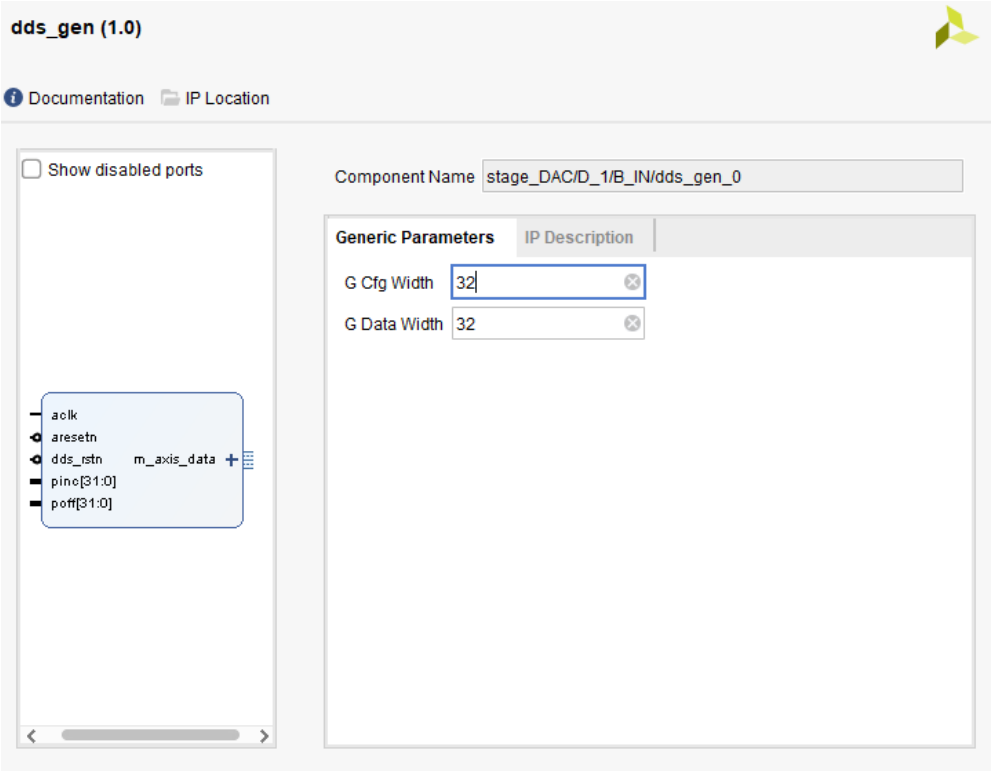

Figura 5-24: Configuración del módulo DDS Compiler

El cálculo para obtener esta resolución de fase (M) se realiza a partir de la Ecuación 6:

$$M = \frac{f_e \times 2^N}{f_s}$$

Ecuación 6: Fórmula de la resolución de fase

Siendo f_s la frecuencia de muestreo, N el número de bits de los que se compone el bus y f_e la frecuencia de la señal buscada.

A la hora de configurar este módulo, es necesario introducir valores iniciales para dichos parámetros. Como la frecuencia de onda no debe sobrepasar la frecuencia límite de Nyquist (encontrada a $f_s/2$), se introduce como valor 2.5 MHz (valor con cierto margen frente a la aparición de *aliasing*).

El valor obtenido de esta resolución de fase es:

$$M = \frac{2.5\ MHz \times 2^{32}}{9.984\ MHz} = 1.07546 \times 10^9$$

Ecuación 7: Cálculo de la resolución de fase introducida por defecto

Por defecto, el valor del *offset* se sitúa con valor “0”.

La introducción de forma externa de estos valores se realiza a partir de GPIOs de salida, donde a través del software se mandan al PL dichos valores. Se debe tener en cuenta que puede aparecer problemas en aspectos de *STA (Static Timing Analysis)* en el diseño. Estos

problemas aparecen cuando se intenta suministrar a partir de una sola GPIO el valor de todos los módulos a los que son destinados en un mismo ciclo de reloj. Para solventarlo, se incluyen una serie de FFs de tal manera que los valores se registran y, aunque llegan ciclos más tarde de reloj, no suponen un problema para la función que están realizando. Aun realizando esta modificación, el valor se establece en diferentes momentos para cada módulo, produciendo la pérdida de sincronización entre señales. Para resolver este problema, se debe de realizar un procedimiento, el cual se basa en detener el flujo a la hora de su configuración. Una vez realizado, se vuelve a reanudar la transmisión.
Esta manera de operar es lo que se realiza a partir del bloque "Control de Flujo" (detallado en el apartado 5.3.5.1.4).

5.3.5.1.1.3 AXI4-Stream Data FIFO

Este módulo se encarga de suministrar muestras al proceso cuando se ha realizado un corte en la transferencia, eliminando la aparición de brechas de datos en la cadena de RF. También sirve para el propósito de conversión de anchos de bus junto al siguiente módulo descrito (AXI4-Stream Data Width Converter).

La configuración establecida se muestra en la Figura 5-25. Los detalles a conocer son la profundidad de almacenamiento (8192) y el ancho de bus que comparte con la DMA (128 bits, es decir, 16 bytes). Respecto al tipo de memoria escogida, se ha dejado por defecto. De esta manera, se va estableciendo los bloques de memoria según interpreta el proceso de implementación.

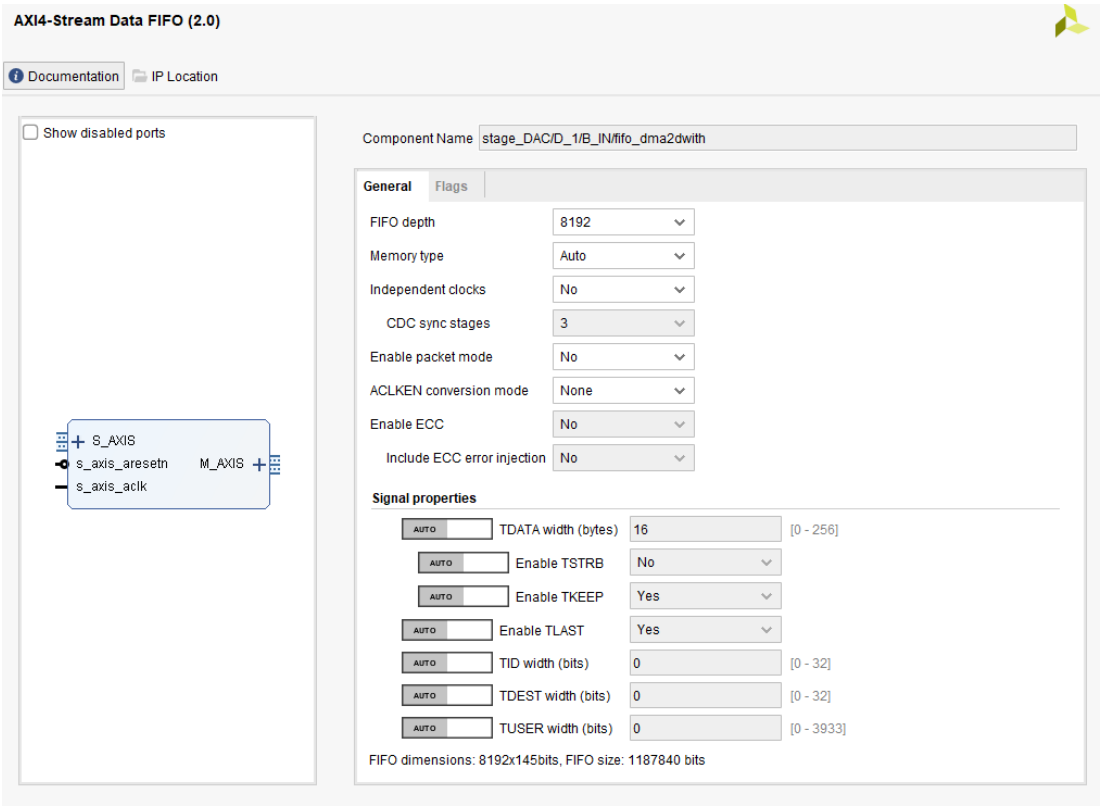

Figura 5-25: Configuración del módulo AXI4-Stream Data FIFO

5.3.5.1.1.4 AXI4-Stream Data Width Converter

Este bloque tiene la funcionalidad de proporcionar el ancho establecido por muestra para este diseño (32 bits). Dependiendo de la conversión que debe realizar, agrega o no latencia en la obtención de datos por salida. La latencia mínima para cualquier proceso es 2. Si la conversión es ascendente, existe aumento de latencia en proporción al tamaño de palabra por salida frente a la entrada. En el caso de una conversión descendente, el proceso se realiza "por goteo" y el orden establecido es por orden de llegada y del bit menos significativo (LSB) al bit más significativo (MSB).

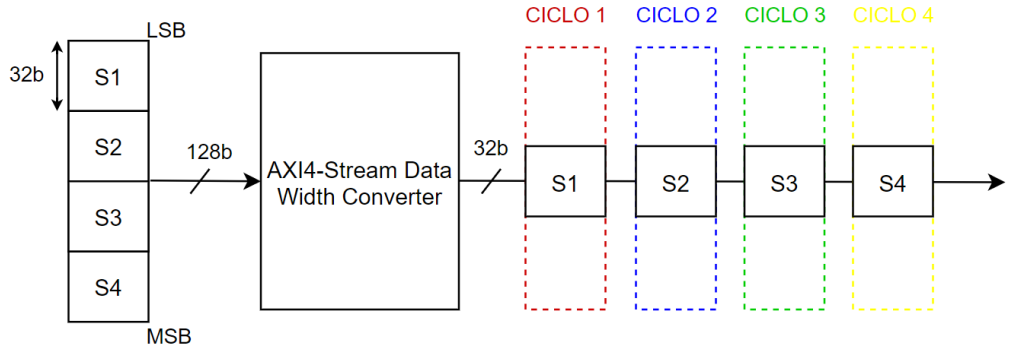

Figura 5-26: Funcionamiento del módulo AXI4-Stream Data Width Converter

La configuración establecida se muestra a continuación. Los detalles por conocer son el tamaño de bus de la interfaz "esclava" 128 bits (proveniente de la FIFO) y el tamaño de bus de la interfaz "maestra" 32 bits.

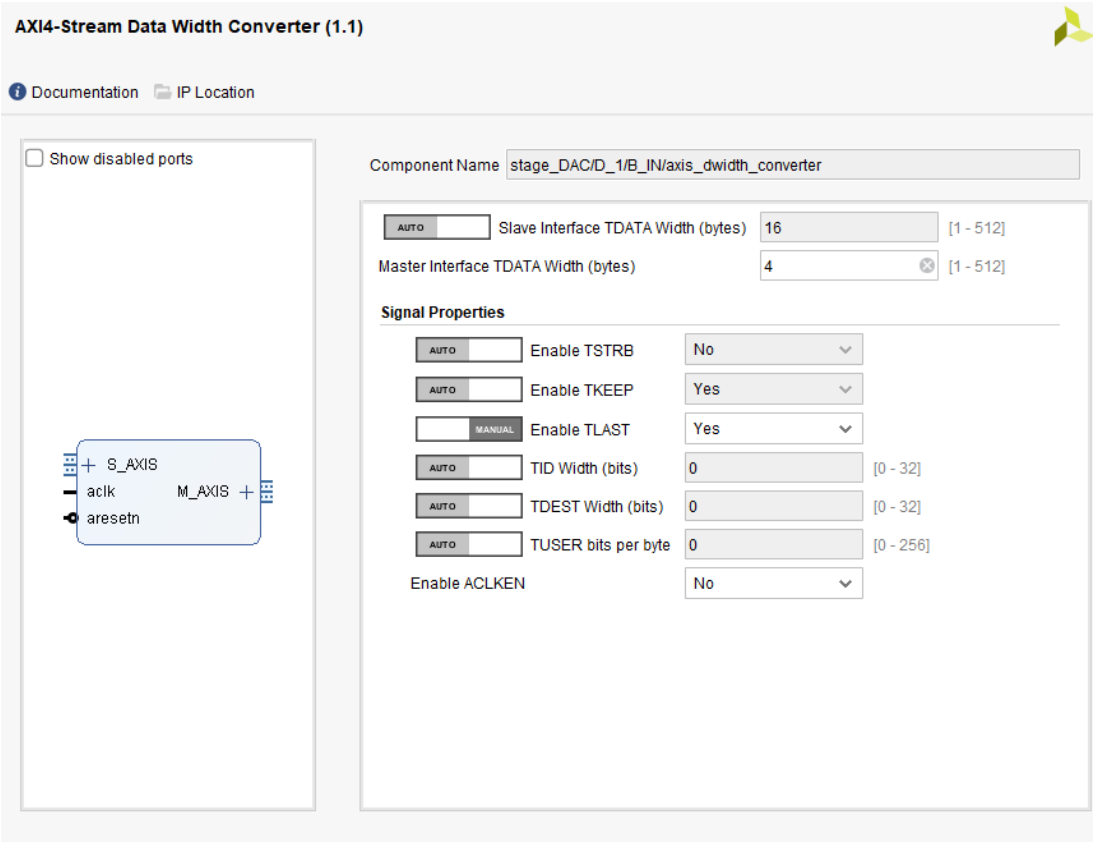

Figura 5-27: Configuración del Módulo AXI4-Stream Data Width Converter

5.3.5.1.1.5 AXI4-Stream Switch

El AXI4-Stream Switch es el módulo encargado de elegir qué volumen de datos se va a introducir en el proceso que lleva a la cadena de RF. El funcionamiento se basa en el enrutamiento configurable entre interfaces "esclavo" y "maestro". El máximo número seleccionable para cada interfaz es 16.
La forma de elegir qué módulo va a tener acceso en este diseño se va a tramitar a través de la opción "*control register routing*". Este registro introduce una interfaz AXI4-Lite para controlar la tabla de enrutamiento.
Las opciones dependen del número de interfaces "esclavas" disponibles. En este caso son 2. Donde la opción "0" sitúa por salida los datos introducidos por la DMA y la opción "1" los datos generados por el módulo "dds_gen".

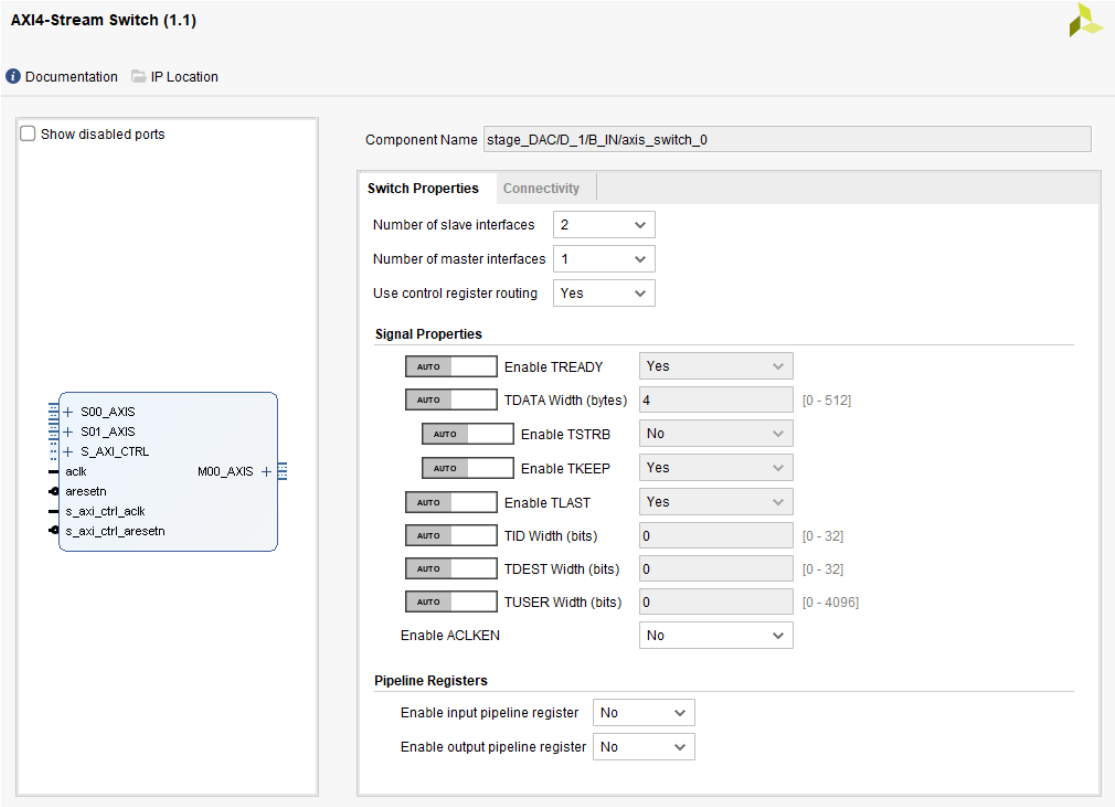

Figura 5-28: Configuración del módulo AXI4-Stream Switch

5.3.5.1.2 Etapas de interpolación

Las etapas de interpolación son el proceso realizado en la cadena para conseguir la reconstrucción de la señal para la obtención de una tasa de muestreo más alta.

Esta conversión de la señal se realiza a partir del módulo FIR Compiler (*Finite Impulse Response*). Dicho módulo proporciona tanto el proceso *Upsampling* como *Downsampling*.

Por los requisitos que se establecen en el diseño, se ha comprobado que para alcanzar tal tasa de muestreo es necesario agregar una etapa de interpolación extra de 40, conociendo que la etapa de interpolación restante (8) se realiza internamente en el módulo RFDC.
Conociendo que este diseño trabaja con varios procesos de interpolación y de diezmado en paralelo, es importante tener en cuenta la carga computacional estipulada en la placa de

evaluación, la cual, equivale al número de DSPs que conforma la RFSoC. En este caso, el número de DSPs contenido es 4272.
Por tanto, es necesario reservar este coste de cómputo a lo largo de todo el proceso. Uno de los principales aspectos para el ahorro en este tipo de conversiones, es dividir la realización de este proceso en subprocesos que realicen la operación con factores más pequeños. Cuanto menor sea el factor que procesar, menor carga computacional interviene. Por consiguiente, si agrupamos todo el conjunto de subbloques en formato serie (en "cascada") equivale a la interpolación total extra que es necesario incorporar en este diseño.

$$Interpolación_{extra} = I_T = 40 = I_1 \times I_2 \times I_3 \times I_4 = 2 \times 2 \times 2 \times 5$$

Ecuación 8: Subfactores de la interpolación extra

Siguiendo esta ecuación, es necesario introducir 4 interpoladores FIR Compiler independientes. Como únicamente existen dos factores distintos (2 y 5), las configuraciones existentes de este módulo se reducen, ya que 3 de estos filtros interpoladores trabajan con el mismo factor (2).

La manera óptima de operar con estos subbloques en la interpolación, es introducir en primera estancia el bloque con menor factor posible, y de aquí los sucesivos. Aunque en este diseño no se ha realizado, la manera más eficiente para eliminar las réplicas es dotar de este primer subbloque con una respuesta en frecuencia de pendiente muy estricta. Esto aumenta los DSPs para este primer subbloque, pero permite a los siguientes filtros interpoladores contener respuestas en frecuencia más laxas, lo cual, estabilizan este crecimiento previo de recursos.

El orden impuesto de los filtros interpoladores es el siguiente: los tres primeros filtros interpoladores operan con un factor 2 y, el último, un factor 5.

Para estipular qué tipo de función deben realizar los FIR Compiler, aparte de la configuración realizada, es necesario introducir una serie de coeficientes. Dependiendo del factor y del tipo de operación (interpolación o diezmado), el conjunto de estos coeficientes aumenta o disminuye. Esto influye también en el número de DSPs necesarios para su implementación. Un alto grado de interpolación se traduce en un gran número de coeficientes.

A la hora de trabajar en Vivado, dichos coeficientes deben de ser introducidos a partir de un archivo con extensión ".coe". Tanto el cálculo de coeficientes como sus valores se pueden encontrar en este documento en el apéndice 9.2. Dichos cálculos son realizados a través de un simple *script* con el programa *Matlab*.

La configuración principal de este módulo es la siguiente:

- En la sección "*Filter Coefficients*" (apartado "*Filter Options"*) se introduce el archivo de coeficientes calculado. En "*Filter Specifications*" se configura el tipo de operación (interpolación o diezmado) y el factor a realizar.

- En "*Channel Specification*" se introduce el número de *paths*, ya que en este módulo se introduce por el mismo bus tanto la parte real como la parte imaginaria, se introduce el valor 2.
  En la sección "*Hardware Oversampling Specificacion*" se ajusta el número de ciclos necesarios para el procesamiento de las muestras. Intentar realizar el cómputo de

forma instantánea afecta directamente en los recursos utilizados. Como en este diseño no es necesaria una actuación tan precisa, para los 3 primeros módulos se ha estipulado 2 ciclos de *clock*, consiguiendo con ello, una utilización de recursos más baja. En el último bloque, a razón de la configuración impuesta en el RFDC, se establece 0.5 ciclos de *clock*, es decir, se obtiene para esta operación de factor 5, 5 muestras en paralelo, aumentando el tamaño de bits que conforman el bus (32 bits x 5 = 160 bits). Esta decisión, aunque se comenta en el módulo RFDC, disminuye la frecuencia necesaria del AXIS en la interacción con el RFDC a costa de introducir más muestras en paralelo.

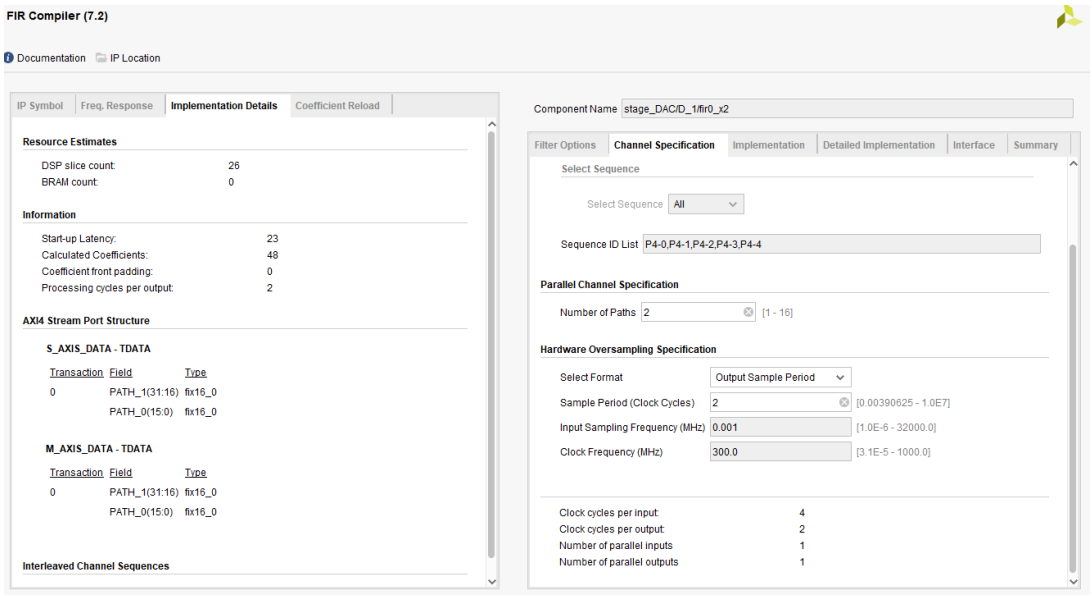

Figura 5-29: "Channel Specification" del módulo FIR Compiler

- En la sección "*Coefficient Options*" (apartado "*Implementation*") se ha establecido la opción de "*Quantize Only*" para que la estructura de los coeficientes sea redondeada al valor más cercano. En *"Coefficientes Width"* se establece el tamaño de palabra de los coeficientes. Aumentar este ancho permite la obtención de mejores resultados en el FIR Compiler, pero implica un mayor coste.
  Una manera de comprobar la eficiencia de estos filtros es comprobar el resultado de la respuesta en frecuencia. Para hacerlo, en la Figura 5-30 se puede visualizar el resultado a partir de la respuesta actual (línea azul) y la respuesta ideal (línea roja), es decir, cuando se establece todo el ancho disponible de todos los coeficientes.
  A medida que se aumenta este ancho se va alcanzando una mejor respuesta. Por tanto, la manera más adecuada de establecer este valor es realizando simulaciones para conocer qué tamaño es el apropiado. En este caso se ha introducido 16 bits.
  Respecto a las demás configuraciones, se ha establecido que tanto la entrada como la salida del filtro sea de 16 bits para seguir estableciendo el mismo formato de 32 bits por muestra. El modo de redondeo está configurado con el formato "*Symmetric Rounding to Zero*", esta opción sirve para redondear y reducir el error introducido al truncar el resultado.

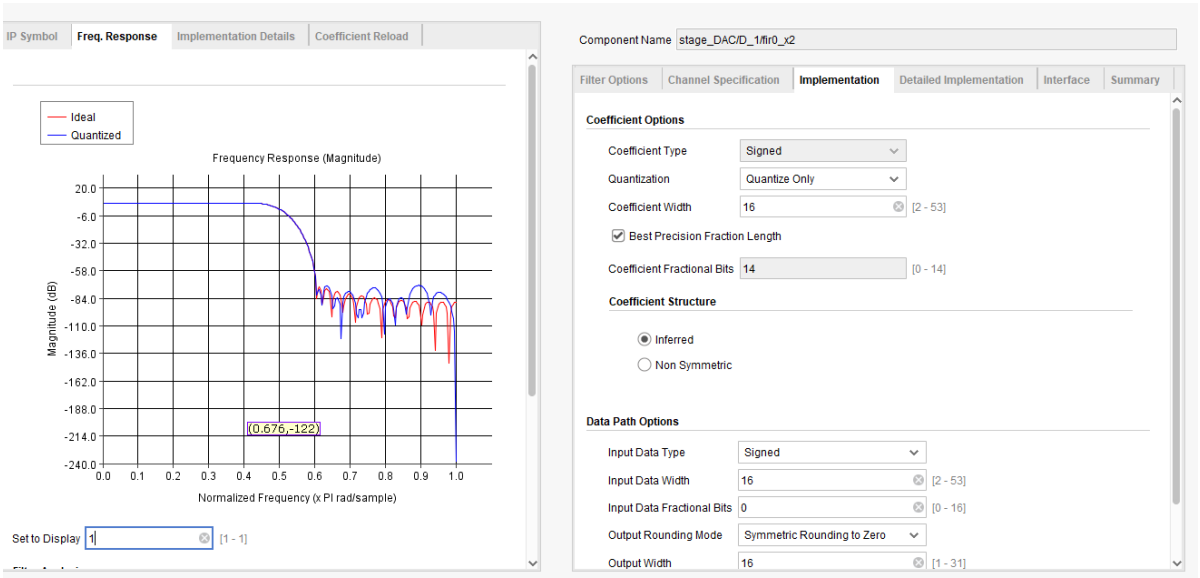

Figura 5-30: "Implementation" del módulo FIR Compiler

- Por último, en el apartado "*Interface*" se ha habilitado para la estructura AXIS la señal *tlast* y *tready*. También se ha incorporado la existencia de una señal reset asíncrono para el proceso.

Para conocer la eficacia de estos filtros interpoladores con sus respectivos coeficientes, se ha trabajado a partir de dos etapas:

- La primera etapa, para ahorrar tiempos en la implementación se han realizado simulaciones previamente con una señal de prueba. Esta señal era introducida por las diferentes etapas de procesado para comprobar si el comportamiento era el adecuado.

- Una vez realizada esta primera etapa y, haber obtenido unos filtros adecuados para el diseño, se ha hecho la etapa de comprobación a nivel de hardware. La placa de evaluación a la que se han implementado estos diseños de prueba ha sido la MPSoC. Por la ausencia de una cadena de RF a la que introducir las muestras de esta señal convertida, se ha trabajado a partir de la monitorización mediante módulos "System ILA" (analizadores lógicos integrados). Estos módulos permiten situar un *trigger* para comprobar anomalías en el comportamiento de la señal obtenida. La profundidad que se ha establecido para monitorizar es de 4096. Y el tipo de interface introducido ha sido "xilinx.com:interface:axis rtl:1.0", ya que el formato a visualizar proviene del bus de datos AXIS.

5.3.5.1.3 AXI4-Stream Data FIFO (CDC)

El AXI4-Stream Data FIFO con formato *CDC*, sigue teniendo una estructura casi idéntica a la comentada en el apartado 5.3.5.1.3. Sin embargo, por la función que este realiza se ha querido definir de forma independiente.

El objetivo de este módulo no es principalmente realizar el almacenamiento, sino el de establecer la transición entre dos dominios de frecuencia. Por ello, una vez habilitado la opción "*Independent clocks*" y "*CDC sync stages*" aparece en su estructura dos frecuencias. El reloj de la interfaz "esclava" es referida al dominio del AXIS (*pl_clk_axis*). Mientras que

el reloj de la interfaz “maestra” es el referente al reloj que interactúa en el apartado de los DACs del RFDC.

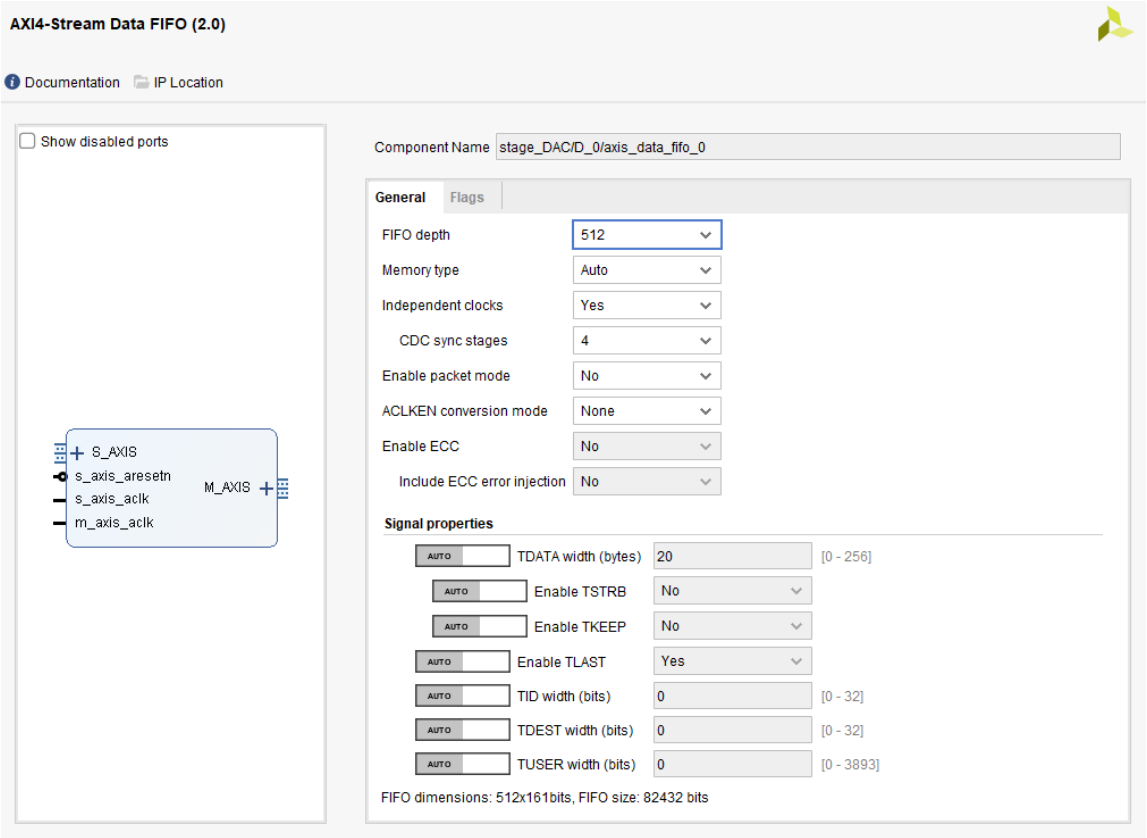

Figura 5-31: Configuración del AXI4-Stream Data FIFO (modo CDC)

5.3.5.1.4 Bloque Control de flujo

El bloque de control de flujo se compone principalmente por dos puertas AND, donde se han añadido dos AXI4-Stream Register Slice para mejorar los resultados de *STA*. La idea es tener el control a manos del diseñador para detener o no la transmisión de datos hacia la cadena de procesado. Para ello, se introduce desde software a través de una GPIO de salida una señal (*control_flujo*) con el valor “1” para que exista comunicación o “0” para cortar la comunicación.

Esto resuelve ciertas situaciones que complican un funcionamiento adecuado, las cuales son:

- La eliminación de brechas al comienzo de la implementación del diseño. Cuando se realiza la instancia del diseño, comienza de forma automática la transmisión de datos en el procesado, el hecho de que estos caminos de datos no contengan un almacenamiento previo, conlleva la aparición de brechas continuas hasta que el flujo se establece de forma completa, es decir, el módulo RFDC obtiene el número de muestras para procesar correctamente.

- Otra situación es a la hora de configurar el diseño a través del software. Los diversos ajustes que varían el funcionamiento de los bloques implicados se realizan a partir de líneas de código. Estas ejecuciones de líneas se realizan una a una. Por tanto, el ajuste establecido no se hace de forma instantánea. Esto conlleva la pérdida de la *multisincronización entre tiles (*MTS*).*

La manera de operar para resolver este problema es bloqueando la salida del procesado (a partir de este control de flujo) causando que las muestras de datos se almacenen en las FIFO, y con ello, la detención de la transmisión. Una vez introducido este ajuste, se levanta el control de flujo para que se vuelva a reanudar la transmisión.

La estructura de este bloque de control de flujo se muestra en la Figura 5-32.

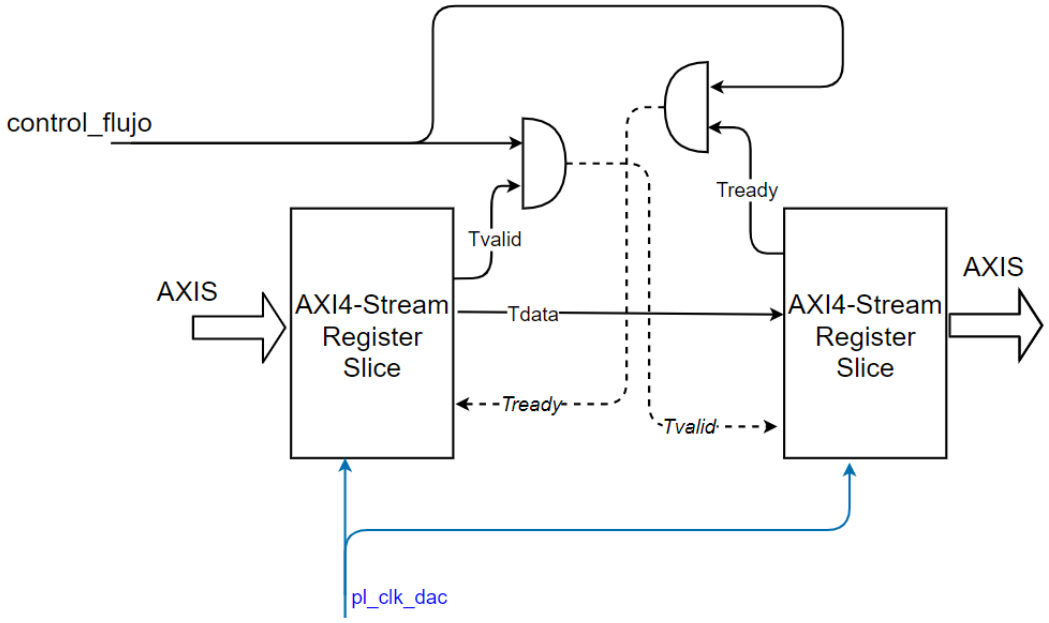

Figura 5-32: Estructura del bloque de "control de flujo"

La funcionalidad trabaja a nivel del protocolo AXIS mediante el conjunto de señales que este comparte. El hecho de estipular con un valor "0" en las puertas AND, origina que sea cual sea el valor de las señales, su resultado es cero. Por tanto, el comportamiento de los *Register Slice* depende de estas señales. En el caso del segundo *Register Slice* le llega la señalización de que el dato de su antecesor no es válido (*tvalid*), por tanto, no existe comunicación. Respecto al primer *Register Slice*, le llega de este segundo, que no está preparado (*tready*) para continuar con la comunicación.

5.3.5.2 Etapa de recepción

La etapa de recepción tiene como objetivo recibir las respuestas del producto a partir de los ADCs de la placa de evaluación ZCU111. Su procesado tiene cierta similitud con el realizado en la etapa de transmisión pero en sentido inverso. En este caso, el volumen de datos procesado va dirigido al PS.

Su estructura se muestra en la Figura 5-33.

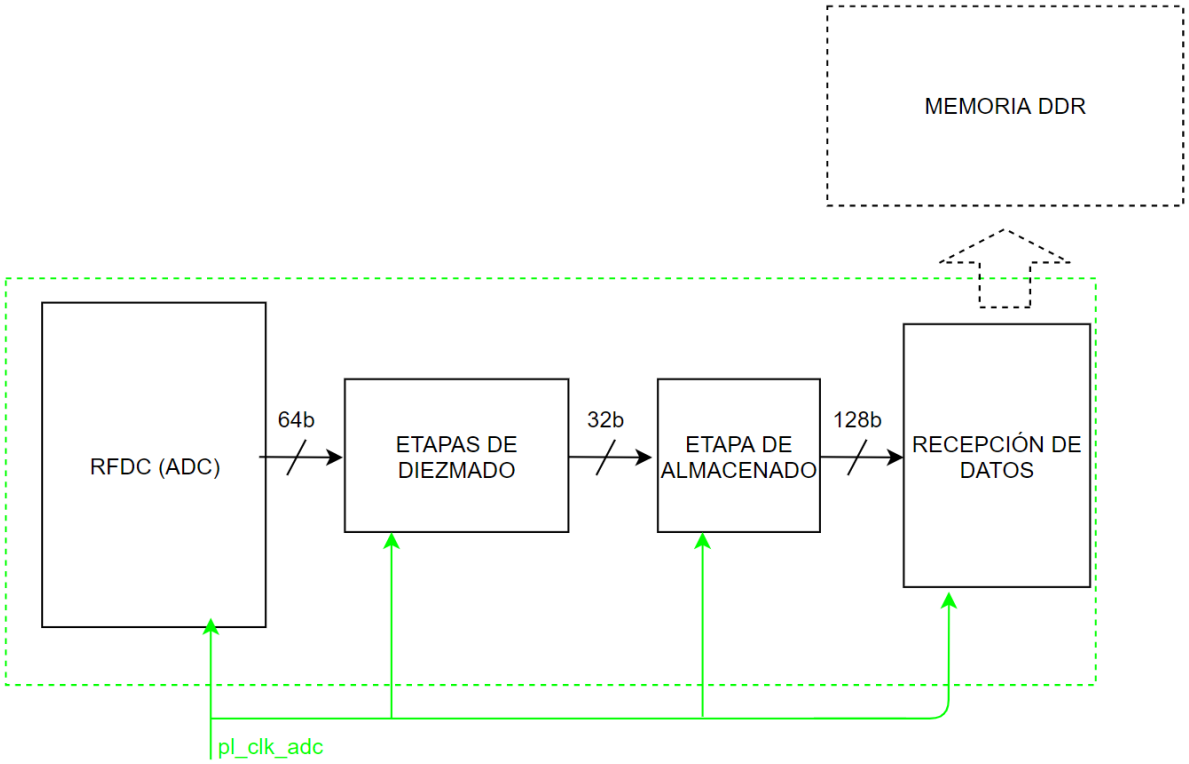

Figura 5-33: Estructura de la etapa de Recepción

Esta estructura trabaja únicamente con un dominio de frecuencia y viene referido al *clock* que necesita el RFDC para procesar los datos que provienen del ADC, siendo este reloj:

- ***pl_clk_adc**:* Tiene un valor de 199.68 MHz. Igual que *pl_clk_dac*, son relojes generados que provienen del LMK.

Antes de describir las configuraciones introducidas para las diferentes etapas. Para no ser redundante, se ha obviado aquellos ajustes que se han utilizado en la etapa de transmisión, mostrando únicamente los más significativos para esta modalidad de recepción.

5.3.5.2.1 Etapas de Diezmado

Debido a la estructura en la que se compone los ADCs, el flujo de datos que circula a través del AXIS es distinto a la forma realizada en el apartado de la etapa de transmisión. Donde esta última, se transmite toda la información por un único bus incluyendo tanto la parte real como la parte imaginaria.
En este caso, por la estructura que el RFDC contiene (DUAL RF-ADC), la parte real y la parte imaginaria circula por distinto bus (16 bits). Esto implica trabajar con el doble de filtros diezmadores para operar ambas partes. Para disminuir este crecimiento extra de carga computacional, se ha disminuido la tasa de transferencia efectiva a lo largo del proceso de diezmado.

A parte de esto último, continua la misma dinámica que la utilizada para los módulos FIR Compiler configurados en la etapa de transmisión. Se sigue utilizando 4 subbloques que realicen el diezmado para factores más pequeños. El orden en este caso en la dirección a la que va el flujo, se estipula que los tres primeros son de factor 2, y el último, de factor 5.

La configuración significativa para estos subbloques es:

- Para el apartado "*Channel Specification*", por el formato del bus AXI que establece el módulo RFDC (con parte I y parte Q independiente), cada filtro diezmador se ha de configurar con "*Number of path*" de valor "1".
  Respecto al número de ciclos para el procesamiento del primer subbloque, se ha introducido el formato "*Out Sample Period*", permitiendo escoger el número de ciclos para operar las muestras de salida. En este caso, se ha seleccionado "1" para realizar la conversión que trae por entrada, obteniendo una única muestra por salida a partir de las dos muestras en paralelo que expulsa el módulo RFDC. Esto establece para las demás etapas de diezmado procesar con el mismo formato. Sin embargo, esta conversión implica un mayor coste. Por ello, se reducirá la carga desechando muestras durante el proceso de diezmado para los siguientes subbloques. La forma de realizarlo es situando el formato "*Input Sample Period*" para aumentar el número de ciclos a la que procesar los datos y, en consecuencia, excluir muestras de forma controlada.
  La configuración para los subbloques siguientes son: Para el segundo subbloque se sitúa el valor 1 obteniendo una tasa de muestras de ½, el tercer subbloque se sitúa el valor 2 obteniendo una tasa de muestras de ¼ y, el último bloque (factor 5), se sitúa el valor 4 obteniendo una tasa de muestras final de 1/20.

  Esta configuración impuesta causa un decrecimiento en el *throughput* del sistema, dichos resultados vienen incluido en el apartado 5.5.

5.3.5.2.2 Etapa de Almacenado

Aunque en esta etapa, el volumen de datos viene suministrada de forma continua a lo largo del tiempo (a diferencia de la etapa de transmisión). En este caso, el *delay* aparece al final del procesado, a partir de la escritura de los datos en el PS.

Para resolver esto, se introducen bloques de almacenamiento AXI4-Stream Data FIFO para que el flujo no se detenga a la hora de tratar con el PS. Estas FIFO contienen una estructura con una profundidad del buffer de 8192. El hecho de no incluir estos módulos de almacenamiento conllevaría pérdidas involuntarias, algo no admitido en este diseño.

Las asignaciones en memoria provocan que se almacenen los datos en la FIFO, por lo cual, es necesario para su vaciado que la salida se transmita con mayor frecuencia o con mayor anchura de bus. Para este caso, se ha escogido la segunda opción. Por ello, seguidamente de la FIFO, se incorpora un AXI4-Stream Data Width Converter que realiza la conversión de 32b a 128b.

Aparte de este almacenamiento, en esta etapa se realiza la conversión a un único bus donde viene incorporado tanto la parte real como la parte imaginaria en su orden respectivo. Para ello, se trabaja a partir del módulo AXI4-Stream Combiner, adquiriendo una salida de 32 bits donde como entrada se introducen dos buses de 16b.

Se debe seleccionar para la interfaz "esclava" con índice "0" el bus correspondiente a la parte real, mientras que para la interfaz "esclava" con índice "1" la parte imaginaria. Esto establece de forma correcta el orden de LSB a MSB.

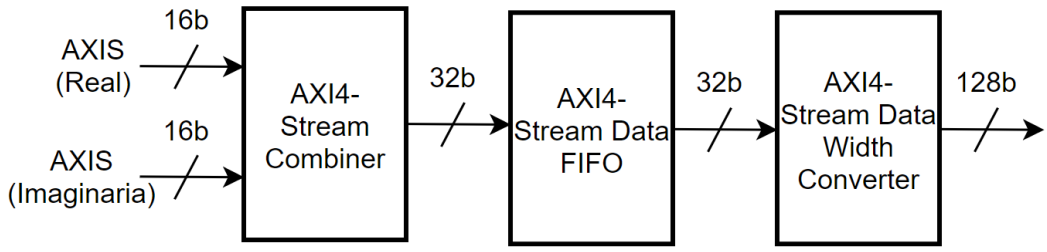

Figura 5-34: Estructura de la etapa de almacenado

5.3.5.2.3 AXI Direct Memory Access (Modo Escritura)

La configuración de la DMA en esta modalidad de recepción es necesario que su estructura tenga habilitado el formato de escritura. De esta manera, los datos obtenidos a lo largo del procesado puedan ser transmitidos del apartado del PL al PS.

Su funcionamiento se basa en la transmisión de datos en memoria DDR a través de su escritura, para ello, se utiliza la interfaz "esclava" (S2MM).
Para incluir esta modalidad se habilita la opción "*Enable Write Channel*". Respecto al tamaño de bus del AXIS, sigue siendo 128 bits y para seguir albergando el mayor *throughput* se introduce "*Max Burst Size*" con el valor 256.

Es importante conocer que la DMA de escritura para realizar esta transición en memoria, necesita conocer qué muestra es la última a lo largo del flujo. La forma de conocerlo es a través de la señal *tlast*. Ya que el RFDC no contiene este funcionamiento de proporcionar dicha señal, se ha introducido un módulo denominado "*tlast_gen*". Su funcionamiento consiste en ir recibiendo y transmitiendo los datos hasta un número de muestras (dicho valor introducido por el diseñador) donde sitúa la subida del flanco de la señal *tlast*. Con esto, la DMA es consciente de cuándo debe de realizar dicha transición.
El valor que estipula el número de muestras donde se debe situar el flanco es introducido a partir de una GPIO.

La configuración del módulo de la DMA con formato de escritura se puede observar en la Figura 5-35.

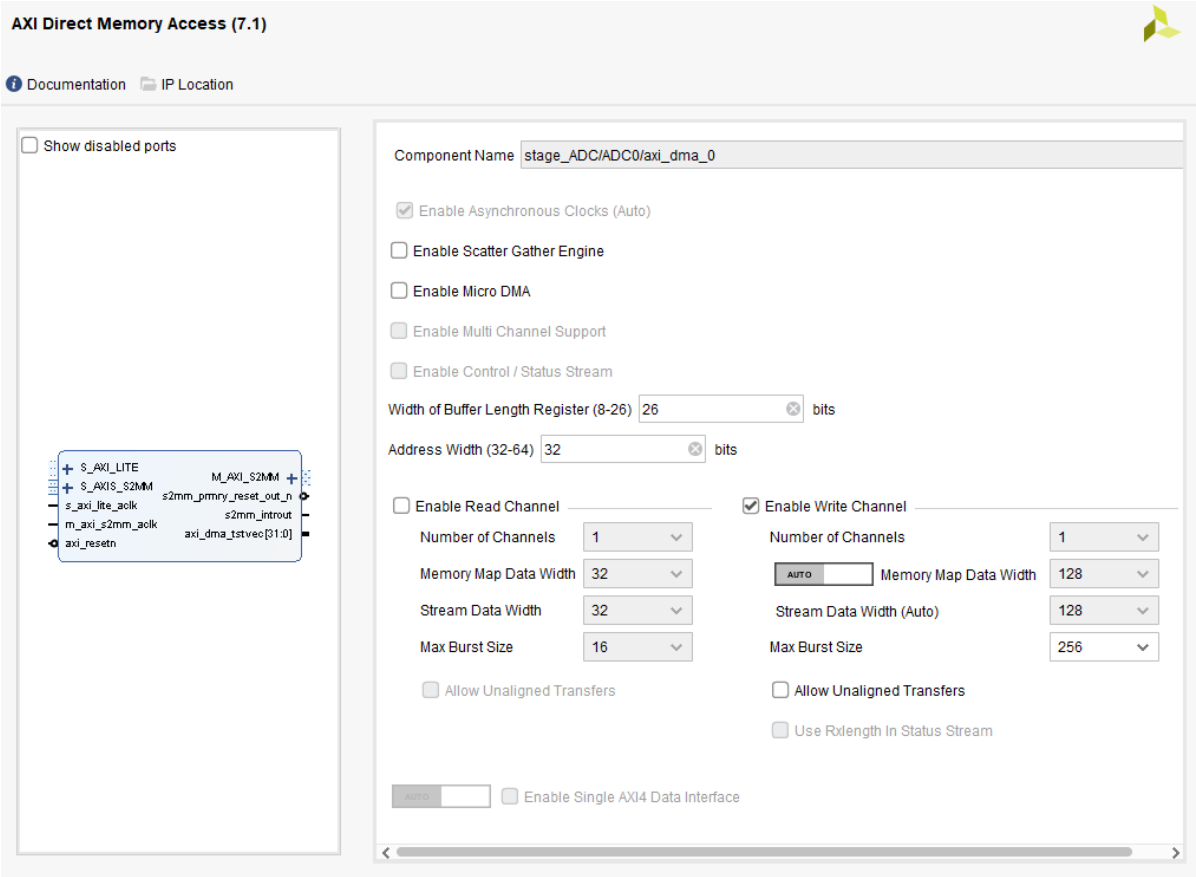

Figura 5-35: Configuración del módulo AXI Direct Memory Access (Modo Escritura)

5.3.6 Zynq Ultra Scale+ RF Data Converter

El módulo RFDC es una IP que integra Xilinx para operar con la cadena de RF de la FPGA (en este caso la RFSoC). Una de las características que trae la cadena de RF, es que vive integrado en el ASIC, lo que origina total accesibilidad a partir buses ARM AMBA-4. Funciona como cualquier IP estándar, la interfaz tanto de los DACs como los ADCs trabaja para la transferencia de datos a nivel del protocolo AXIS y tan solo para la configuración previa de registros a través del AXIL.

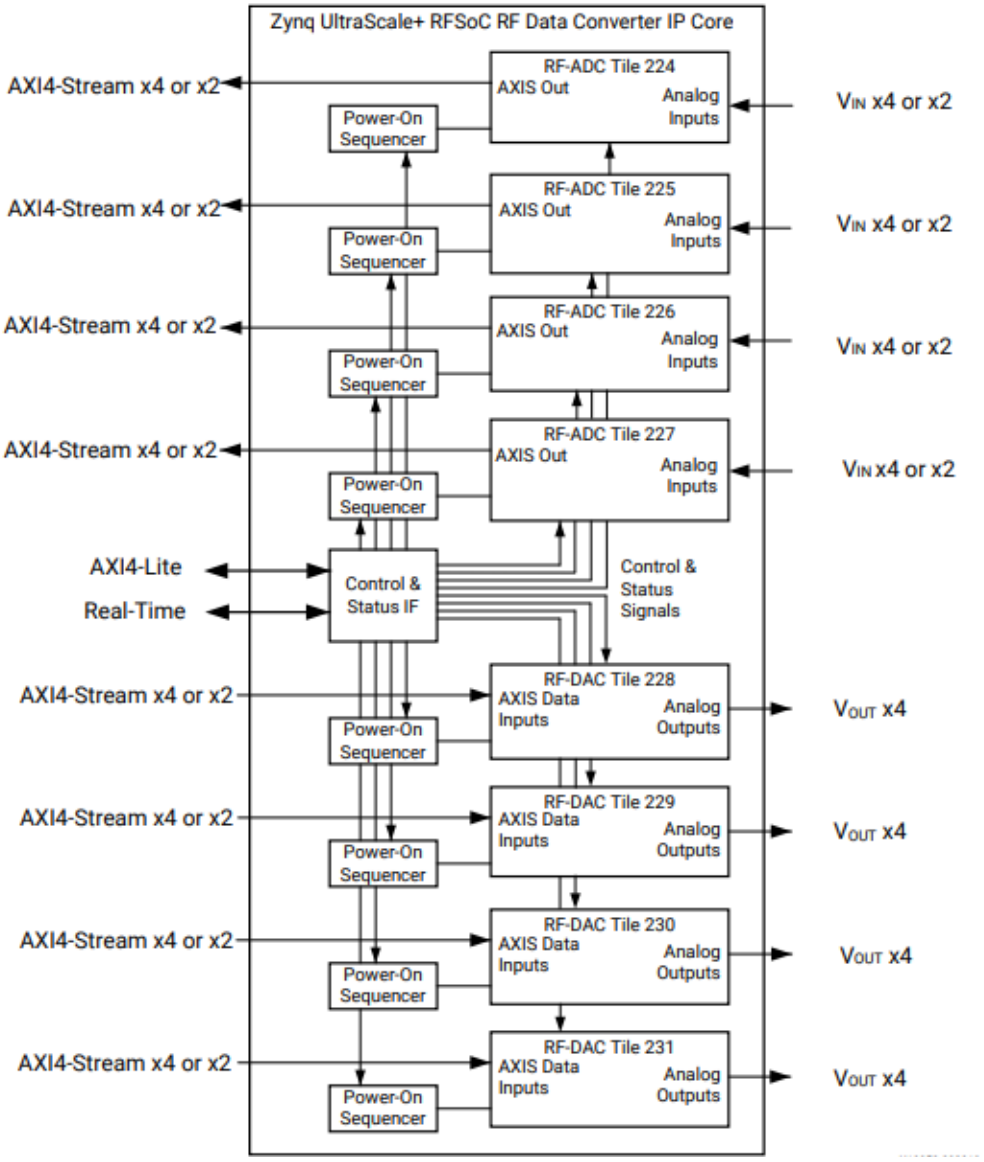

Figura 5-36: Estructura del módulo Zynq Ultra Scale+ RF Data Converter [19]

Siguiendo la descripción de esta IP, la forma de organización de esta cadena es a partir de *tiles*. Esto es debido a la compartición de recursos comunes para conseguir, si fuera preciso, la "coherencia" entre *tiles* mediante la sincronía de relojes.

Dependiendo de la placa de evaluación manejada, el número de componentes y *tiles* puede variar. Para este caso, la RFSoC contiene 2 *tiles* de 4 DACs cada uno. Y por el otro lado, existen 4 *tiles* donde vienen integrados dos ADCs.
Para ambas partes, la ZCU111 contiene 8 elementos en total (16 elementos si contamos ADCs y DACs).

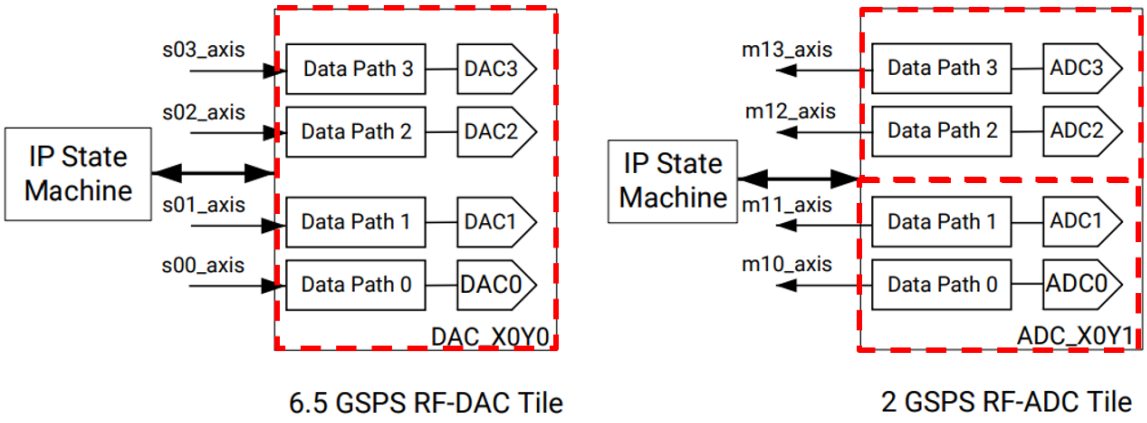

Figura 5-37: Estructura de los *tiles* (DACs y ADCs) [20]

La metodología que tiene esta placa para administrar los datos a los DACs/ADCs proviene de los transceptores GTY.

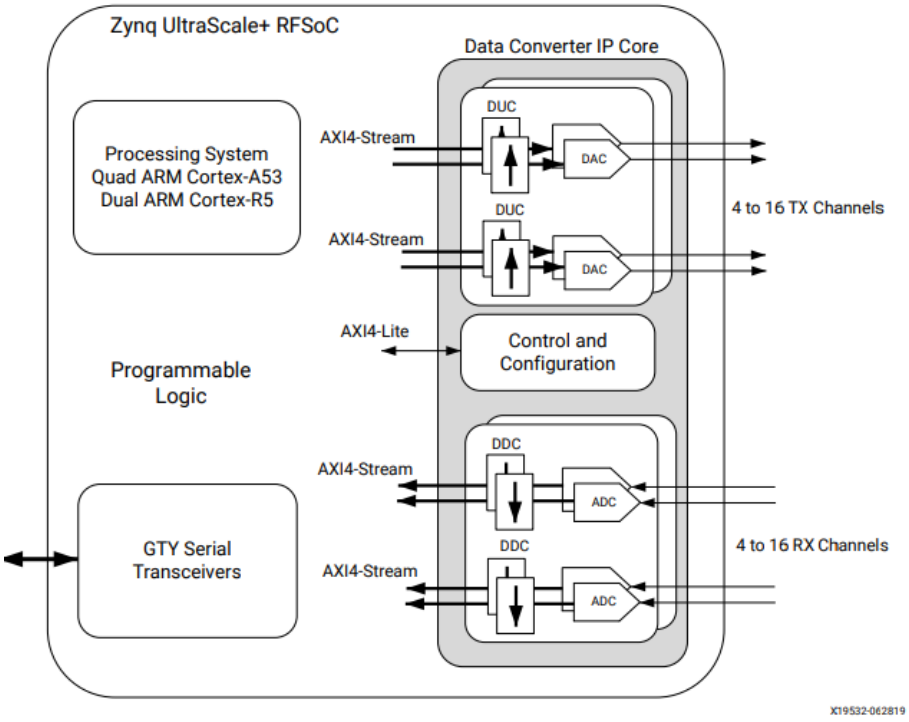

Figura 5-38: Comunicación del RFDC [21]

Una limitación que trae consigo, es la instanciación exclusiva de esta IP para poder operar con esta cadena de RF en el apartado del PL.

5.3.6.1 Conversión de señal digital a analógica - DAC

Las características de los DACs de esta placa son:

- Soporte de AXIS (AXI4-Stream).
- Resolución RF-DAC de 14 bits.
- Tasa de muestreo hasta 6.554 GSPS.

- Filtros de interpolación de 1 (*bypass*), 2, 4 y 8.
- Conjunto de mezcladores complejos digitales para la salida de señales reales o I/Q (complejas). Además de Osciladores (NCO) de 48b.
- Elección de una/multi banda (depende de la configuración elegida para la salida de la señal).
- 80% de ancho de banda en la zona de Nyquist y atenuación en la banda "stopband" de 89 dB.
- Señal de entrada externa "*sysref"* para la sincronización multicanal.
- Modo de potencia: salida de intensidad de 20 mA o 32mA.

La función de los DACs a partir del mezclador interno es modular una señal desde banda base (BB) hasta una frecuencia portadora. Además de realizar la interpolación de los datos entrantes.

La metodología para el procesado de señal comienza con el almacenado de la señal a partir de muestras en una FIFO (se encuentra internamente en el RFDC). A partir de aquí, la señal se introduce por diferentes etapas:

- **Etapa de interpolación**: Para realizar la etapa de interpolación, la señal es procesada por un convertidor digital ascendente (DUC). Está formado por un "*Upsampler*" donde se elige el grado de interpolación (1, 2, 4 u 8), seguido de un oscilador (NCO) que mediante el mezclador incorporado, modula la señal. Cabe destacar que si el grado de interpolación es 1 (*bypass*), el proceso de *Upsampling* no se implementa. También se ha de añadir que en esta etapa no existe ningún filtro *antialiasing* para la eliminación de réplicas (esto se verá afectado en la señal resultante).

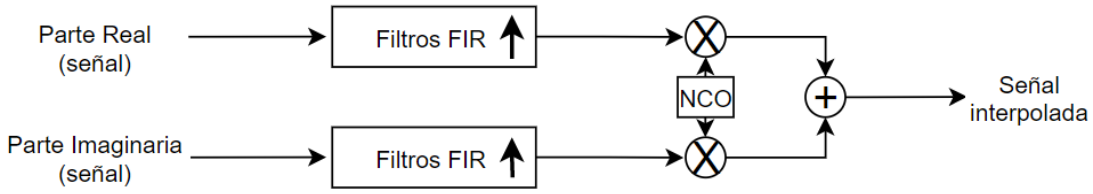

Figura 5-39: Procedimiento de "Upsampling"

- La configuración del mezclador contiene dos opciones: "Mezcla fina" (*Fine Mixing*) para modular a una frecuencia arbitraria o "Mezcla gruesa" (*Coarse Mixing*), la cual permite mezclar a partir de una frecuencia portadora teniendo las siguientes opciones: $0, \pm F_s/4, F_s/2, F_s$, siendo F_s la frecuencia de muestreo.

- La etapa del QMC (Corrector de modulador cuadrático) se utiliza para corregir los errores inducidos del modulador cuadrático de la cadena DAC. Estos errores aparecen debido a la corriente continua (CC), la ganancia y la fase, los cuales originan errores espectrales en forma de espurio.

El diagrama de bloques del dispositivo DAC se muestra en la Figura 5-40.

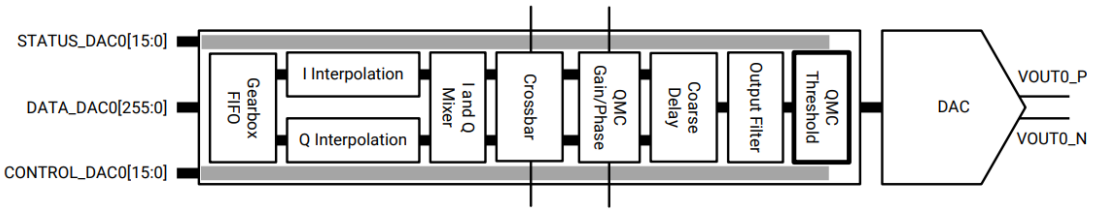

Figura 5-40: Estructura de los componentes DAC [22]

5.3.6.2 Conversión de señal analógica a digital – ADC

Las características de los ADCs son:

- Soporte de AXIS (AXI4-Stream).
- Resolución RF-ADC de 12 bits.
- Tasa de muestreo de hasta 4.096 GSPS.
- Filtros de diezmado de 1 (*bypass*), 2, 4 y 8.
- Conjunto de mezcladores complejos digitales para la entrada de señales reales o I/Q (complejas). Además de Osciladores (NCO) de 48b.
- Elección de una/multi banda (depende de la configuración elegida para la salida de la señal).
- 80% de ancho de banda en la zona de Nyquist y atenuación en la banda "*stopband*" de 89 dB.
- Señal de entrada externa "*sysref"* para la sincronización multicanal.

La función de los ADCs, a partir del mezclador interno, es demodular la señal portadora hasta la frecuencia en banda base (BB). Además de realizar el proceso de diezmado.

La configuración del NCO es idéntica al apartado de los DACs.

Respecto al proceso del diezmado, la principal diferencia está en el uso de un DDC en sustitución al DUC.

- **Etapa de diezmado:** El procedimiento se realiza a partir de un convertidor digital descendente (DDC). Está formado por un mezclador (mediante el uso de un oscilador), quien demodula la señal hasta la frecuencia en banda base y, por último, un *"Downsampler"*.

  Igual que en el apartado de DACs, contiene una FIFO, pero esta vez al final de todo el proceso.

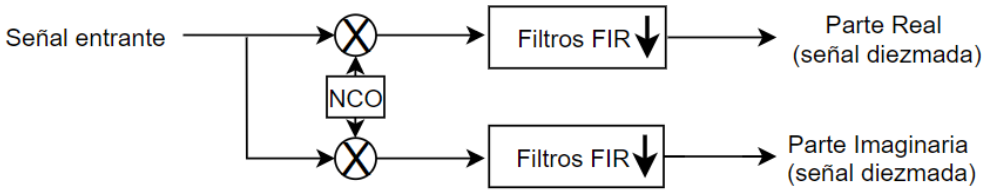

Figura 5-41: Procedimiento de "Downsampling"

En esta placa de evaluación (ZCU111), los ADC no procesan tanto la muestra real como la imaginaria por el mismo bus de datos AXIS. Esto es debido a la estructura implementada

"DUAL RF-ADC" (Figura 5-42). No sería el caso si dicha estructura fuera "QUAD RF-ADC", donde se puede introducir ambas muestras por el mismo bus.

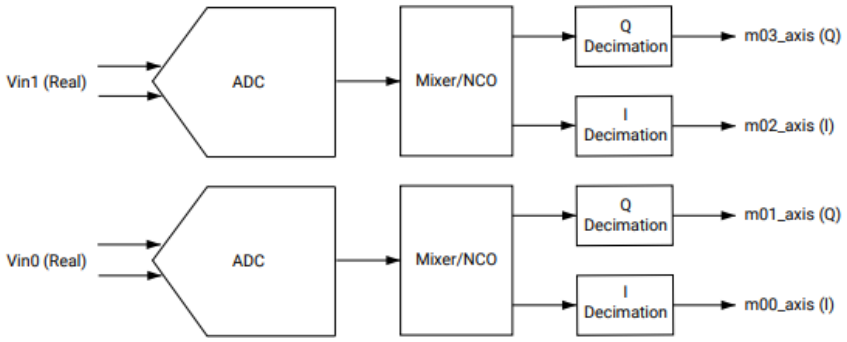

Figura 5-42: Estructura del componente ADC (Dual RF-ADC) [23]

El diagrama de bloques del dispositivo ADC se muestra en la Figura 5-43.

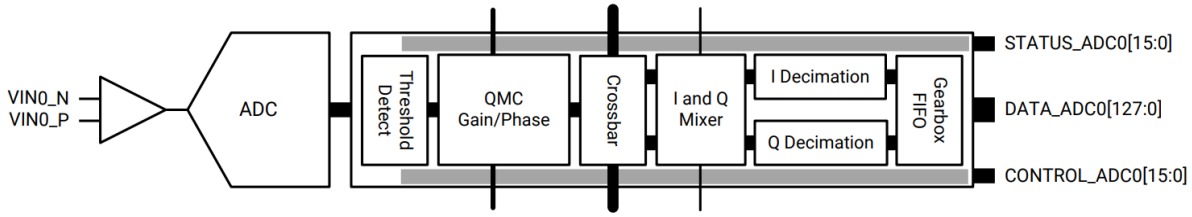

Figura 5-43: Estructura de los componentes ADC [24]

5.3.6.2.1.1 Configuración del módulo Zynq Ultra Scale+ RF Data Converter

A través de la configuración del módulo RFDC, se conocen las frecuencias que se deben generar a través del *Clocking Wizard* en la etapa de generación de *clocks*. Viene en relación por la tasa de muestreo escogida (3.19488 GSPS). A partir de ahí, comienza la configuración de la manera más adecuada a la hora de operar con este módulo para las etapas de Tx y Rx.

- La configuración comienza habilitando en el apartado "*Basic*" el número de componentes que se van a utilizar en el diseño (tanto DACs como ADCs). Una vez hecho, en el apartado "*System Clocking*" se introduce la tasa de muestreo que se va a utilizar.
  Existen dos maneras para obtener esta frecuencia para la tasa de muestreo. La primera a través de los sintetizadores LMXs y la segunda a través de un PLL que viene integrado en este módulo. Se ha optado por la primera opción.
  También en este apartado se puede observar el máximo alcance de la tasa de muestreo que procesan ambos componentes, además de las frecuencias que necesitan para la realización del procesamiento de muestras. El bus AXIS de los DACs se introduce en la interfaz "esclava", mientras que el bus de los ADCs se introduce en la interfaz "maestra".
  El valor de estas frecuencias se encuentra en la tabla "*Fabric Clocks*" de la Figura 5-44.

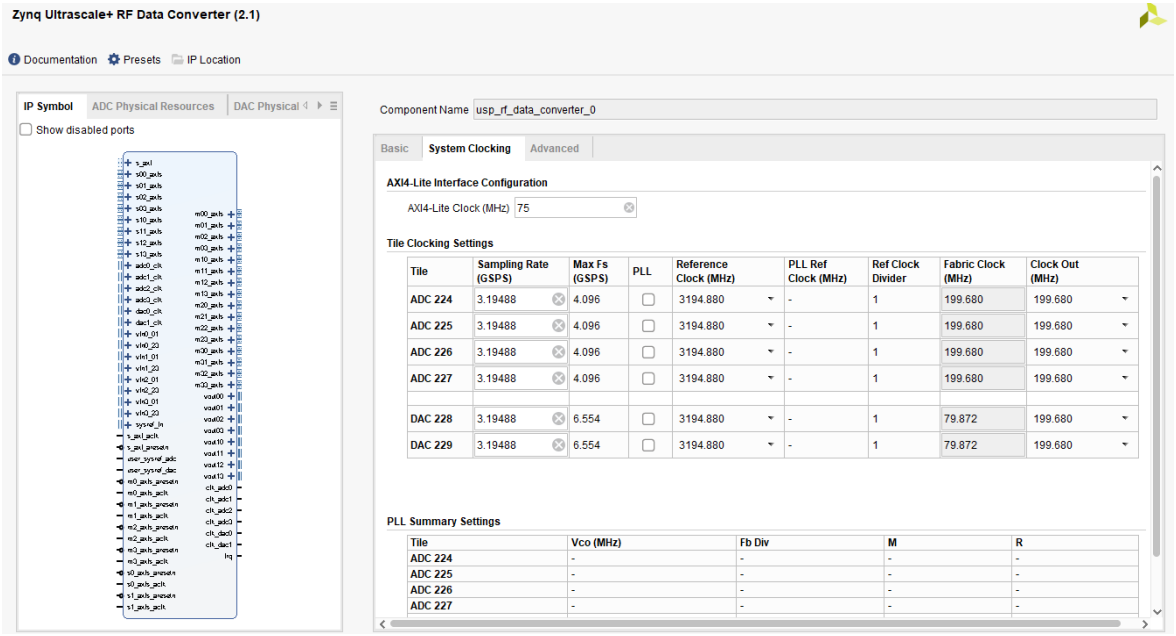

Figura 5-44: "System Clocking" del Módulo Zynq Ultra Scale+ RF Data Converter

Estas frecuencias que estimulan las interfaces del AXIS tanto de los ADCs como los DACs, pueden variar con relación al aumento o disminución de muestras en paralelo que se procesanr. Esto también influye en el bus de datos que circula por el PL. Para modificar estos detalles, de nuevo hay que seleccionar el apartado "*Basic*".

- Una vez seleccionado, en el apartado DACs se elige como tipo "*Real*" el dato a transmitir para la salida analógica. El factor de interpolación se configura asignando el mayor valor (8). El número de muestras en paralelo se establece como 10 (conteo total de muestras entre reales e imaginarias). En el aspecto del mezclador utilizado, se ha configurado como tipo "*Fine*" para poder escoger cualquier frecuencia portadora a la que modular la señal. El modo del mezclador se ha seleccionado la opción "*IQ→Real*", ya que las muestras que se procesan son complejas. Y, por último, tanto la zona Nyquist como el "*Decoder mode*" se dejan por defecto, siendo estas opciones la "*Zona 1*" y "*SNR Optimized*" respectivamente.

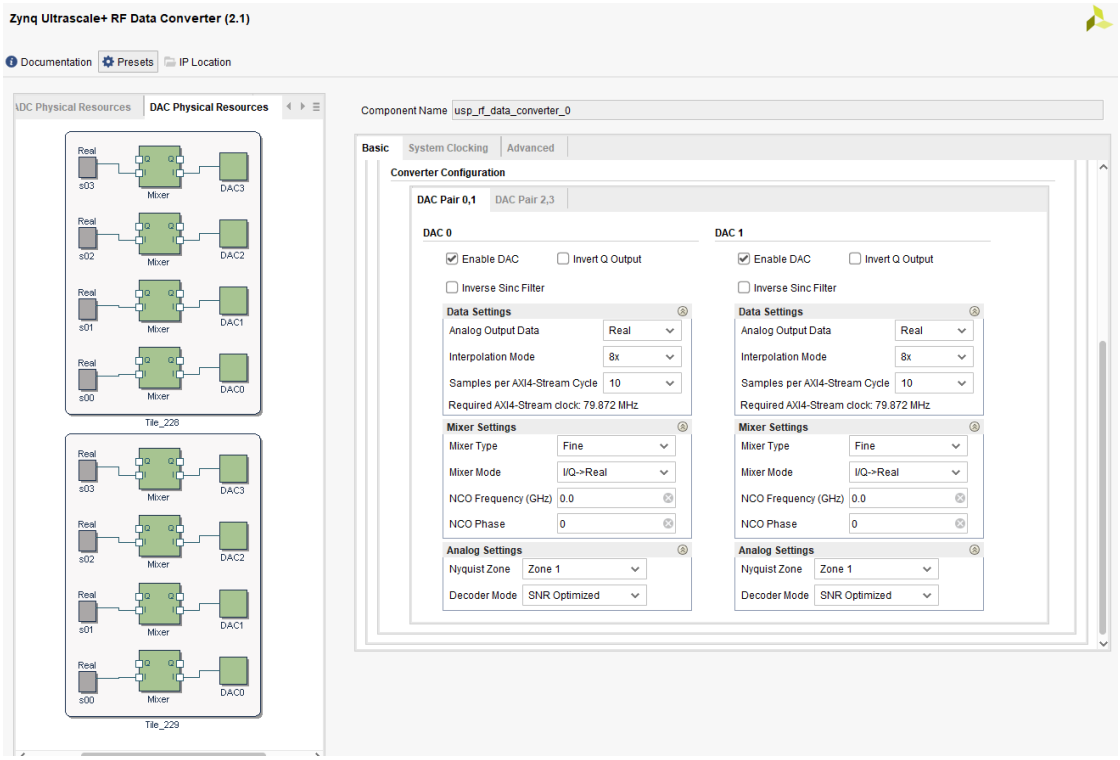

Figura 5-45: "Basic" del módulo Zynq Ultra Scale+ RF Data Converter (DACs)

- Respecto al apartado de los ADCs, se establece como formato de salida analógica de tipo "*I/Q*", para obtener por el par de buses de datos AXIS las muestras obtenidas reales e imaginarias. El diezmado establecido se introduce con el factor 8. El número de muestras en paralelo se configura con el valor 2. El tipo de mezclador también se introduce el formato "*Fine*". Y, por último, la configuración "*Analog Setting*" se vuelve a dejar por defecto.

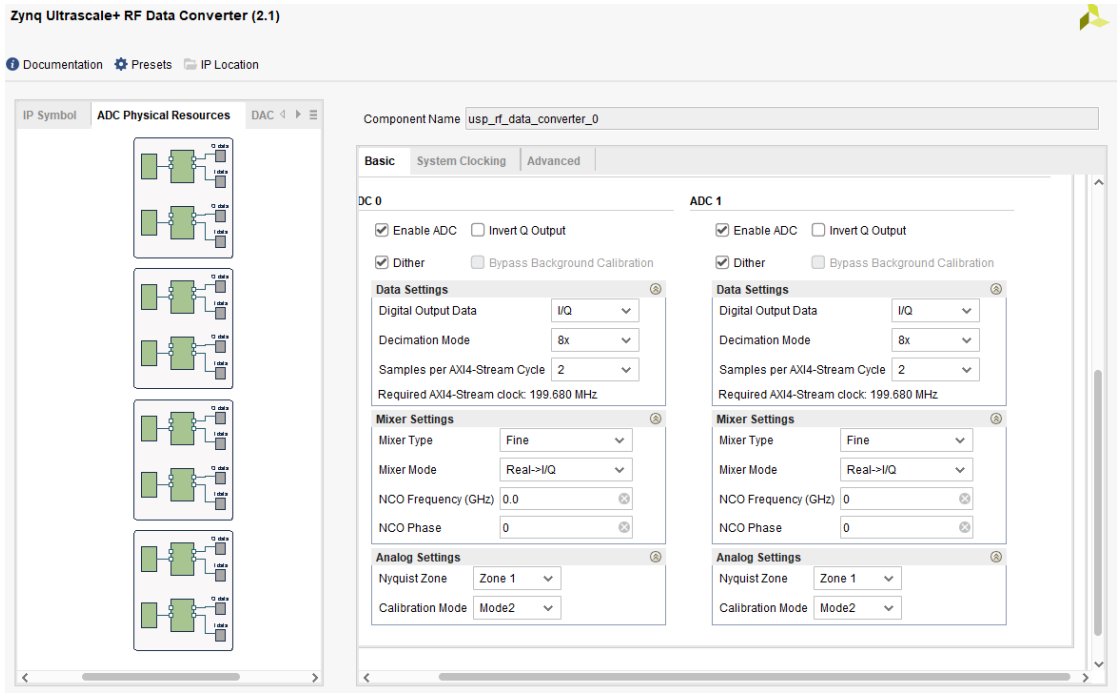

Figura 5-46: "Basic" del módulo Zynq Ultra Scale+ RF Data Converter (ADCs)

Con estas configuraciones, se obtienen las frecuencias de *pl_clk_dac (79.872MHz)* y *pl_clk_adc (199.68 MHz)*. Dichas frecuencias vienen generadas a partir del módulo Clocking Wizard.

Funcionalidad de la *multisincronización entre tiles*
Es necesario para este diseño, una vez configurado ambos apartados, habilitar la opción "*Enable Multi Tile Sync*". Esto añade a la estructura del módulo las entradas de la señal SYSREF (*user_sysref_dac* y *user_sysref_adc)* para obtener la sincronización entre canales.

Con esto se termina la creación del diseño, una vez cerrado el *Block Design*, se crea el *wrapper*, es decir, el archivo de VHDL que rodea toda la lógica del circuito y donde se establecen tanto las entradas como las salidas del diseño.

El siguiente paso es la etapa de crear el archivo de restricciones ("*constrains*").

5.4 Aplicación de los archivos de "Constrain"

Una vez validado el diseño, se realiza de forma opcional el apartado "*RTL analysis*" para conocer realmente los nombres que estipula Vivado para referirse tanto a las entradas como a las salidas del diseño.

Este apartado se basa en implementar las restricciones de enrutamiento (*placement*) y de tiempo (*timing*).

5.4.1 Archivo constrain de "*Placement*"

Este archivo consiste en enrutar las señales de *clock* generadas por loss sintetizadores externos, siendo estas frecuencias: PL_CLK y SYSREF.
Visualizando la ficha "*Squematic*" [25] que proporciona Xilinx, se buscan los pines correspondientes a estos relojes.

Los pines correspondientes se encuentran en la Figura 5-47.

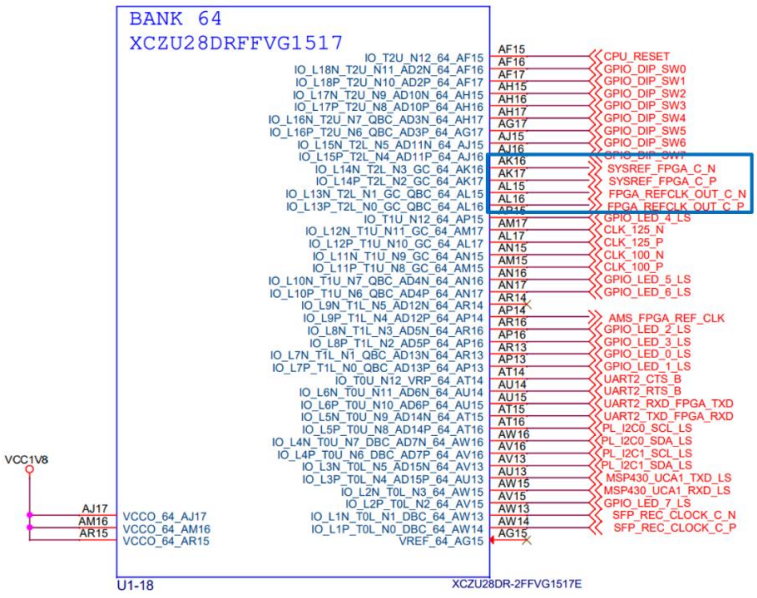

Figura 5-47: Esquemático ZCU111 (pines SYSREF y PL_CLK) [25]

Por ser pines diferenciales se incluyen ciertas propiedades para completar su configuración.

El archivo *constrain* de *Placement* es el que se muestra a continuación:

```
set_property PACKAGE_PIN AL16 [get_ports {PL_CLK_P_N_clk_p[0]}]
set_property PACKAGE_PIN AL15 [get_ports {PL_CLK_P_N_clk_n[0]}]
set_property PACKAGE_PIN AK17 [get_ports {SYSREF_P_N_clk_p[0]}]
set_property PACKAGE_PIN AK16 [get_ports {SYSREF_P_N_clk_n[0]}]

set_property IOSTANDARD LVDS [get_ports {PL_CLK_P_N_clk_p[0]}]
set_property IOSTANDARD LVDS [get_ports {PL_CLK_P_N_clk_n[0]}]
set_property IOSTANDARD LVDS [get_ports {SYSREF_P_N_clk_p[0]}]
set_property IOSTANDARD LVDS [get_ports {SYSREF_P_N_clk_n[0]}]

set_property DIFF_TERM_ADV TERM_100 [get_ports {PL_CLK_P_N_clk_p[0]}]
set_property DIFF_TERM_ADV TERM_100 [get_ports {SYSREF_P_N_clk_p[0]}]

set_property DQS_BIAS TRUE [get_ports {PL_CLK_P_N_clk_p[0]}]
set_property DQS_BIAS TRUE [get_ports {PL_CLK_P_N_clk_n[0]}]
set_property DQS_BIAS TRUE [get_ports {SYSREF_P_N_clk_p[0]}]
set_property DQS_BIAS TRUE [get_ports {SYSREF_P_N_clk_n[0]}]
```

5.4.2 Archivo constrain de "Timing"

Este archivo tiene como objetivo instanciar las frecuencias que se van a introducir de forma externa de tal manera que la placa de evaluación se adecue a la hora de la implementación. Los aspectos de tiempo vienen referidos a:

- **PL_CLK_P_N**: Con frecuencia 122.88 MHz
- **SYSREF_P_N**: Con la frecuencia calculada para este caso, 3.072 MHz.

La manera de introducirlo se realiza escribiendo el período (en nanosegundos) equivalente a ambas frecuencias.

```
create_clock -period 8.138 -name pl_clk [get_ports {PL_CLK_P_N_clk_p[0]}]
create_clock -period 325.521 -name pl_sysref [get_ports {SYSREF_P_N_clk_p[0]}]
```

En la guía del RFDC se añade cierto *delay* para que la SYSREF no se incorpore en ciclos donde pueda producirse "metaestabilidad".

Para conocer estos valores de *Tsetup* y *Thold*, se debe introducir por comando tcl: "*report_datasheet".*

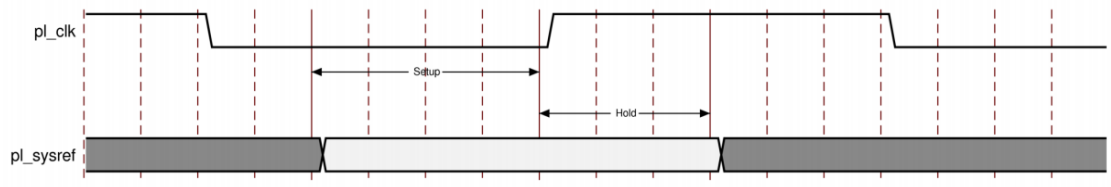

Figura 5-48: Delay para el archivo "constrain" de "Timing" [26]

5.5 Cálculos de la tasa de transferencia efectiva (throughput)

Antes de comenzar el apartado software, se muestran a continuación los cálculos de la tasa de transferencia efectiva obtenido a lo largo de todo el diseño. Dicho análisis vienen establecidos una vez conocidos los anchos de buses y las frecuencias de trabajo. Siendo estas frecuencias:

- *pl_clk_axis*: 250 MHz
- *pl_clk_dac*: 79.872 MHz
- *pl_clk_adc*: 199.68 MHz

Etapa de Transmisión:

- En la etapa de generación de datos, la transferencia entre PS-PL entre la DMA alcanza:

  250 MHz x 128 bits = 32 Gbps

  (siendo para 8 *DACs*, se incrementa 8 veces, es decir, 256 Gbps)

- Sin embargo, este flujo pasa por un convertidor de ancho de 128 bits a 32 bits (formato similar al módulo DDS Compiler), reduciéndose la tasa un cuarto de su valor:

  250 MHz x 32 bits = 8 Gbps

- Esta tasa se mantiene hasta el último filtro interpolador extra, el cual aumenta el ancho de bus de 32 bits a 160 bits:

  250 MHz x 160 bits = 40 Gbps

- En la etapa de la FIFO (CDC) existe un cambio de dominio de frecuencias del *pl_clk_axis* al *pl_clk_dac*:

  79.872 MHz x 160 bits = 12.779 Gbps

- Estos 12.779 Gbps se introducen en el RFDC y se procesa, por último, la etapa final de interpolación. Obteniendo:

  12.779 [Gbps] x 8 / 32 [bits/muestra] = 3.19475 GSPS

  Cuyo valor es la tasa de muestreo escogida (3.19488 GSPS).

Etapa de Recepción:

- El módulo RFDC tiene establecido una tasa de muestreo de 3.19488 GSPS. Internamente se realiza un diezmado de factor 8 de la señal, obteniendo una tasa de transferencia efectiva de:

  3.19488 [GSPS] x 32 / 8 [muestra / bits] = 12.779 Gbps

  De igual manera se puede calcular de esta manera, donde se obtienen del módulo RFDC dos muestras de 32 bits a 199.68 MHz, es decir, una muestra a 399.36 MHz:

  399.36 MHz x 32 bits = 12.779 Gbps

- En la primera etapa de diezmado se disminuye de dos muestras a una en paralelo:

  199.68 MHz x 32 bits = 6.38976 Gbps

- En la segunda etapa se desechan la mitad de las muestras, es decir, ½:
  199.68 MHz x 32 bits x 1 / 2 = 3.19488 Gbps

- En la tercera etapa se desecha de nuevo la mitad de las muestras, obteniendo ¼:
  199.68 MHz x 32 bits x 1 / 4 = 1.59744 Gbps

- Por último, con el objetivo de realizar el vaciado de la FIFO (por los problemas que acontecen la escritura en memoria), existe una conversión de ancho de bus de 32b a 128b. Obteniendo como tasa final:

199.68 MHz x 128 bits x 1 / 20 = 1.277952 Gbps, es decir, 9.984 MHz x 128 bits

6 Configuración del Software

Una importante apreciación que contiene esta placa es que es una de las pocas tarjetas oficiales que cuenta con la compatibilidad del *framework* PYNQ. Se trata de un nuevo enfoque a la hora de implementar el desarrollo de una aplicación, de modo que la capacidad completa del SoC y la placa están disponibles de forma sencilla a través de la programación mediante *Python*.
Para entender mejor las bases en las que se mueve PYNQ, se proporciona el anexo B.

6.1 Objetivos del Software

Los principales objetivos del software son los siguientes:

- Integración y conversión del conjunto de datos a tratar que sea compatible con la estructura implementada en el diseño (hardware). Se debe de tener en cuenta la especificación de la memoria RAM contenida en la tarjeta ZCU111.
- Configuración de los principales módulos que intervienen en el diseño: GPIOs, DMAs, RFDC, switch, etc.
- Implementación de los relojes externos al PL (generados por LMX y LMK).
- Implementación de la funcionalidad "MTS" ("coherencia entre canales" tanto en DACs como ADCs).

6.2 Metodología de PYNQ

6.2.1 Generación de archivos

El funcionamiento del entorno PYNQ requiere la generación del *bitstream* del diseño (".bit"). También, es esencial copiar el archivo ".hwh" (o su forma análoga a partir de comandos tcl) donde se dispone la estructura del diseño. También a partir de dicho archivo se puede conocer los diferentes bloques que intervienen en el diseño con su respectivo nombre (algo muy intuitivo a la hora de programar).

6.2.1 Comunicación con PYNQ y transmisión de archivos

Una vez encendida la placa con la imagen cargada, es necesario acceder a la dirección IP de la RFSoC (por defecto, la dirección IP es *192.168.2.99*). La comunicación entre el ordenador y la placa se conecta a través de un cable Ethernet.
Una vez todo configurado, se ejecuta el server de *Notebook Jupyter*, donde debe incluirse los archivos requeridos mencionados anteriormente (".bit" y ".hwh")

6.2.2 Desarrollo de aplicaciones mediante *Jupyter Notebooks*

Gracias a esta herramienta de Python se puede crear de forma instantánea aplicaciones sobre el navegador.

El proceso es el siguiente:

- Empleo de las librerías requeridas para el funcionamiento del diseño.
- Declaración e inicialización de los diseños.
- Implementación de la funcionalidad buscada.

6.2.3 Implementación del diseño en apartado Software

Este apartado tiene como objetivo integrar la funcionalidad requerida a partir de instrucciones y operaciones para el diseño integrado.

6.2.3.1 Importación de librerías

En primer lugar, se importa el *Overlay* (el objeto principal donde se introduce el *core* del diseño, es aquí, la conversión de la información incluida del *bitstream* al formato *Python*).

```
from pynq import Overlay
overlay = Overlay("archive_bitstream.bit")
```

El siguiente paso es la integración de todas las librerías en las que se hace referencia el *Overlay*. En este diseño, es necesario incluir tanto la librería *xrfclk* como *xrfdc* (ambas requeridas para el módulo RFDC).

```
import xrfclk
import xrfdc
```

- La librería **xrfclk**, cuyo *driver* "__init__.py" permite la configuración de los relojes de referencia para dicha placa. Estos relojes se configuran a partir de una serie de registros. Para ello, es necesario operar a partir de un programa (creado por la empresa *Texas Instruments*) llamado TICS Pro. Él se encarga de programar módulos de evaluación (EVM) de terminología CDC, LMK y LMX (dispositivos donde se incluyen PLLs y osciladores). La configuración utilizada para este programa se encuentra en el anexo A.

- Respecto a la librería ***xrfdc,*** también con su *driver* "__init__.py", contiene lo referido a la estructura interna del RFDC. De modo que para configurar sus opciones es necesario utilizar las "clases" de su estructura. En esta librería también se ha incluido funciones para obtener la "coherencia entre canales" (principalmente *runMTS(), sysrefDisable()* y *sysrefEnable()*).

Para modificar el contenido de estas librerías es necesario acceder a partir de un terminal por medio del comando "*ssh xilinx@192.168.2.99*".
Una vez dentro, se accede a cada librería mediante su respectivo *path*. Siendo xrfdc (/usr/local/lib/python3.6/dist-packages/xrfdc) y xrfclk (/usr/local/lib/python3.6/dist-packages/xrfclk).

Aparte de importar estás librerías "base", se han introducido algunas extras para poder operar con otras funcionalidades como son: la realización de cálculos matemáticos, visualización de gráficas y reservado de memoria. Esta última es necesario a la hora de trabajar con los módulos DMAs, ya que funcionan mediante un *buffer* que requiere de almacenado de memoria para poder operar.

```
import numpy as np
import matplotlib.pyplot as plt
from pynq import allocate
```

Otras librerías incluidas están relacionadas con los módulos implementados en el diseño (DMA y GPIOs):

```
from pynq.lib import AxiGPIO
import pynq.lib.dma
```

6.2.3.2 Configuración del RFDC

Para instanciar el módulo en el diseño se introduce el *path* donde este se encuentra.

```
rf = overlay.usp_rf_data_converter_0
```

Para introducir la tasa de muestreo elegida, la serie de registros que configuran los sintetizadores (LMX y LMK) debe incluirse en la librería *xrfclk*. Para implementarlo en el diseño, se debe introducir la siguiente función:

```
xrfclk.set_all_ref_clks(3194.88)
```

Está función comprueba si el conjunto de registros de la tasa de muestreo (para el LMX) se encuentra actualmente en la librería. En el caso de que no fuera así, devuelve un código error. Cabe destacar que esta función no permite "por defecto" configurar el LMK desde la aplicación. Sin embargo, se puede adecuar dicha función para que pueda ser proporcionada.

```
def set_all_ref_clks(freq):
    """Set all RF data converter tile reference clocks to a given frequency.
    freq: Target frequency, as selected from `get_freq_list()`.
    """
    if not freq in _lmx2594Config:
        raise RuntimeError(f"Frequency of {freq} MHz is not an option. "
                           "Please see available options in "
                           "getFreqList()")
    else:
        _safe_wrapper("writeLmk04208Regs", _iic_channel, _lmk04208Config[122.88])
        _safe_wrapper("writeLmx2594Regs", _iic_channel, _lmx2594Config[freq])
```

Dependiendo de la configuración establecida en el hardware, se habilita el conjunto de DACs/ADCs, siguiendo la estructura que contiene la ZCU111 (2 *tiles* para DACs y 4 *tiles* para ADCs).

La implementación en PYNQ para la integración de estos dispositivos es la siguiente:

```
#### DAC tiles 1 & 2
dac_tile0 = rf.dac_tiles[0]
dac_tile1 = rf.dac_tiles[1]
#DACs
#tile 0
dac00 = dac_tile0.blocks[0]
dac01 = dac_tile0.blocks[1]
dac01 = dac_tile0.blocks[2]
dac01 = dac_tile0.blocks[3]
#tile 1
dac10 = dac_tile1.blocks[0]
dac11 = dac_tile1.blocks[1]
dac12 = dac_tile1.blocks[2]
dac13 = dac_tile1.blocks[3]

#### ADC tiles 1 & 2 & 3 & 4
adc_tile0 = rf.adc_tiles[0]
adc_tile1 = rf.adc_tiles[1]
adc_tile2 = rf.adc_tiles[2]
adc_tile3 = rf.adc_tiles[3]

#ADCs
#tile 0
adc00 = adc_tile0.blocks[0]
adc01 = adc_tile0.blocks[1]
#tile 1
adc02 = adc_tile1.blocks[0]
adc03 = adc_tile1.blocks[1]
#tile 2
adc10 = adc_tile2.blocks[0]
adc11 = adc_tile2.blocks[1]
#tile 3
adc12 = adc_tile3.blocks[0]
adc13 = adc_tile3.blocks[1]
```

6.2.3.2.1 Configuración del Mezclador del módulo RF Data Converter

Una de las características de esta cadena de procesado, es la posibilidad de modular y demodular la señal procesada.

Dependiendo del tipo de evento (*Event Source*) operado en el mezclador. Existen tres opciones:

- **EVNT_SRC_IMMEDIATE**: Para la modulación o demodulación a través de un evento inmediato, sin tener en cuenta ningún tipo de sincronización (ni en mismo *tile* ni distinto *tile*).
- **EVNT_SRC_TILE**: Permite realizar la operación teniendo en cuenta la sincronización entre componentes del mismo *tile.*

- **EVENT_SRC_SYSREF**: Permite conseguir la funcionalidad buscada *MTS* (sincronización entre mismo y distinto *tile*).

La configuración introducida viene en relación con el hardware impuesto. Esto quiere decir que, aunque por software se puede modificar pequeños ajustes, las principales configuraciones vienen incorporadas por el hardware. En este diseño, solo se ha modificado desde el software algunas opciones del mezclador a la hora de operar con la señal.

A continuación, se muestra la configuración utilizada para el mezclador del RFDC, para no repetir el mismo proceso para cada componente, se ha definido la estructura de configuración para el primer DAC/ADC del primer *tile*. El parámetro "*freq"* es el valor de la frecuencia portadora a la que modular/demodular la señal.

- Apartado DAC

```
dac00.NyquistZone = 1
dac00.MixerSettings = {
    'CoarseMixFreq':  xrfdc.COARSE_MIX_BYPASS,
    'EventSource':    xrfdc.EVNT_SRC_IMMEDIATE,
    'FineMixerScale': xrfdc.MIXER_SCALE_1P0,
    'Freq':           freq,
    'MixerMode':      xrfdc.MIXER_MODE_C2R,
    'MixerType':      xrfdc.MIXER_TYPE_FINE,
    'PhaseOffset':    0.0
}
dac00.UpdateEvent(xrfdc.EVENT_MIXER)
dac_tile0.SetupFIFO(True)
```

- Apartado ADC

```
adc00.NyquistZone = 1
adc00.MixerSettings = {
    'CoarseMixFreq':  xrfdc.COARSE_MIX_BYPASS,
    'EventSource':    xrfdc.EVNT_SRC_TILE,
    'FineMixerScale': xrfdc.MIXER_SCALE_1P0,
    'Freq':           freq,
    'MixerMode':      xrfdc.MIXER_MODE_R2C,
    'MixerType':      xrfdc.MIXER_TYPE_FINE,
    'PhaseOffset':    0.0
}
adc00.UpdateEvent(xrfdc.EVENT_MIXER)
adc_tile0.SetupFIFO(True)
```

Para la obtención de la "coherencia entre canales", es necesario seguir el procedimiento de la guía del RFDC ("*MTS Sequencing*") [29].
Los pasos que se deben realizar son los siguientes:

1. Desactivar la señal SYSREF.
2. Configuración del Mezclador (*Mixer*).
3. Reset de los NCO.
4. Habilitar la señal SYSREF.
5. Inhabilitar la señal SYSREF.

La estructura para este caso es:

- Apartado DAC

```
rf.sysrefDisable()
dac00.NyquistZone = 1
dac00.MixerSettings = {
   'CoarseMixFreq':  xrfdc.COARSE_MIX_BYPASS,
   'EventSource':    xrfdc.EVNT_SRC_SYSREF,
   'FineMixerScale': xrfdc.MIXER_SCALE_1P0,
   'Freq':           freq,
   'MixerMode':      xrfdc.MIXER_MODE_C2R,
   'MixerType':      xrfdc.MIXER_TYPE_FINE,
   'PhaseOffset':    0.0
}
dac_tile0.SetupFIFO(True)
dac00.ResetNCOPhase()
rf.sysrefEnable()
rf.sysrefDisable()
```

- Apartado ADC

```
rf.sysrefDisable()
adc00.NyquistZone = 1
adc00.MixerSettings = {
   'CoarseMixFreq':  xrfdc.COARSE_MIX_BYPASS,
   'EventSource':    xrfdc.EVNT_SRC_SYSREF,
   'FineMixerScale': xrfdc.MIXER_SCALE_1P0,
   'Freq':           freq,
   'MixerMode':      xrfdc.MIXER_MODE_R2C,
   'MixerType':      xrfdc.MIXER_TYPE_FINE,
   'PhaseOffset':    0.0
}
adc_tile0.SetupFIFO(True)
adc00.ResetNCOPhase()
rf.sysrefEnable()
rf.sysrefDisable()
```

Todo lo que viene referido a las funciones que realizan la funcionalidad de *MTS*, no vienen implementadas en las librerías de PYNQ. Por ello, es necesario incluirlas. Estas funciones se encuentras referenciadas en la guía del RFDC en lenguaje *baremetal* (es necesario su conversión al formato *Python)*. Dichas funciones se han introducido en el archivo "*__init__.py*" de la librería xrfdc (en la clase del módulo RFDC).

```
def RunMTS(self):
    self._DAC_Sync_Config = _ffi.new("XRFdc_MultiConverter_Sync_Config*")
    self._ADC_Sync_Config = _ffi.new("XRFdc_MultiConverter_Sync_Config*")
    self.entero = _ffi.new("int*")
    self._DAC_Sync_Config.Tiles = 0x3
    self._DAC_Sync_Config.SysRef_Enable = 0x0
    self._ADC_Sync_Config.Tiles = 0xf
    self._ADC_Sync_Config.SysRef_Enable = 0X0
    _lib.XRFdc_MultiConverter_Init(self._DAC_Sync_Config,self.entero,self.entero)
    _lib.XRFdc_MultiConverter_Init(self._ADC_Sync_Config,self.entero,self.entero)
    self.status1 = _lib.XRFdc_MultiConverter_Sync(self._instance,_lib.XRFDC_DAC_TILE,sel
f._DAC_Sync_Config)
    self.status2 = _lib.XRFdc_MultiConverter_Sync(self._instance,_lib.XRFDC_ADC_TILE,sel
f._ADC_Sync_Config)

    self.enablePtr = _ffi.new("u32*")
    self.enable =  _ffi.new("u32*")
```

```
def sysrefDisable(self):
    self.status3 = _lib.XRFdc_MTS_Sysref_Config(self._instance, self._DAC_Sync_Config, s
elf._ADC_Sync_Config, 0)
```

```
def sysrefEnable(self):
    self.status3 = _lib.XRFdc_MTS_Sysref_Config(self._instance, self._DAC_Sync_Config, s
elf._ADC_Sync_Config, 1)
```

6.2.3.3 Configuración para la etapa de transmisión

Para los siguientes módulos configurados es necesario instanciar su estructura mediante el *path* donde se encuentran respecto al *Overlay*.

6.2.3.3.1 Configuración del switch

Las opciones del *switch* son “0x0” para la elección de la DMA o “0x1” para el módulo DDS Compiler (viene establecido por el orden situado en el módulo AXI-Stream Switch para las interfaces “esclavas” del hardware). El parámetro que escoge dicha opción se denomina “*sel*”.

```
sel=0x1
#switch DDS/DMA DMA->0, DDS->1
switch_dds_dma10 = overlay.stage_DAC.D_0.B_IN.axis_switch_0
sw_offsetMaster = 0x40 # dirección del maestro
switch_dds_dma10.write(sw_offsetMaster, sel)
switch_dds_dma10.write(0x0,0x2)
```

Para el resto de *paths* se debe de configurar específicamente para cada caso, situando D_0 para configurar el *datapath* del DAC0, D_1 para el DAC1, etc.

6.2.3.3.2 Configuración de la DDS

La forma de configurar la DDS se realiza a través de dos GPIOs, suministrando el parámetro *pinc* y el parámetro *poff.*

```
gpio_dds_00_pinc = overlay.ip_dict["stage_DAC/axi_gpio_pinc"]
dds_00_pinc = AxiGPIO(gpio_dds_00_pinc).channel1
gpio_dds_00_poff = overlay.ip_dict["stage_DAC/axi_gpio_poff"]
dds_00_poff = AxiGPIO(gpio_dds_00_poff).channel1
```

Para conocer el valor de la resolución de fase (*pinc*), se calcula a través de su función (mostrada en la Ecuación 6).

```
def DdsCalcPhaseIncrement(freq):
    c = (pow(2, 32)) / (9.984*1000000)
    return (int) (c * (freq * 1000000))
```

Para introducir dicho valor en el PL, se escribe sobre una GPIO a través de la función *nombre_gpio.write(valor, máscara).*

```
pincVal = DdsCalcPhaseIncrement(freq)
dds_00_pinc.write(pincVal,mask)
```

6.2.3.3.3 Configuración de la DMA (Modo “lectura”)

Para el caso de la DMA se debe inicializar cada bloque con su respectiva señal de interrupción.

```
DMA_TX = overlay.stage_DAC.D_1.B_IN.axi_dma_TX
interrupt_dma00 = DMA_TX.mm2s_introut
```

También es necesario reservar en memoria el *buffer* que contiene todo el volumen de datos perteneciente al PS. Como en el diseño se está trabajando con muestras de 32 bits se utiliza el formato *int32.*

```
in_buffer = allocate(shape=(n,), dtype=np.int32)
```

Las dos funciones principales para la DMA son:

- *sendchannel.transfer(X)*: Es la función que envía los datos al PL, siendo “X” el buffer de entrada.
- *sendchannel.wait()*: Esta función asegura que la transición ha tenido éxito. Para ello, se realiza la espera necesaria para que se complete el proceso.

A continuación, se muestran ejemplos de los diferentes escenarios utilizados en este proyecto.

6.2.3.3.3.1 Escenario: Señal sinusoidal equivalente a un tono

La generación de una señal sinusoidal a través de este módulo se realiza a partir de la función "*sine_gen*". Sus argumentos son: amplitud "*a*", frecuencia central en MHz "*f*", frecuencia de muestreo en MHz "*fs*", componente phi en radianes "*phi*" y el número de muestras que forma la señal "*n*".

También es necesario la función *to_fp(data)* para realizar la debida conversión de los datos al formato del diseño, donde se localiza la parte real en los primeros 16 bits menos significativos y la parte imaginaria en los 16 bits más significativos. Además de realizar si fuera necesario su extensión de signo.

```
n=16*1024
def sine_gen(a, f, fs, phi, n):
    w = 2 * np.pi * f
    dt = 1 / fs
    t = np.arange(0, n, 1) * dt
    xt = a * np.exp(1j*w*t + phi)
    return [xt, t, w]

def DMA_Sine(freq):
    a = 1
    f = freq*1000000
    fs = 9.984*1000000
    phi = 0
    signal_gen = sine_gen(a, f, fs, phi, n)
    array_fp = to_fp(signal_gen[0])
    for i in range(n):
       in_buffer[i] = array_fp[i]

def to_fp(data):
    data_real = np.multiply(np.real(data), np.power(2, 15)).astype(np.int32)
    data_imag = np.multiply(np.imag(data), np.power(2, 15)).astype(np.int32)
    data_real = np.where((data_real >= 32768), 32767, data_real)
    data_imag = np.where((data_imag >= 32768), 32767, data_imag)
    data_iq = np.zeros(len(data_real), dtype=np.uint32)
    data_imag_shifted = np.uint32(np.bitwise_and((data_imag << 16), 0xFFFF0000))
    data_real_shifted = np.uint32(np.bitwise_and((data_real << 0), 0x0000FFFF))
    data_iq = np.bitwise_or(data_imag_shifted, data_real_shifted)
    return data_iq

DMA_Sine(freq)
while(1<2):
    DMA_TX.sendchannel.transfer(in_buffer)
    DMA_TX.sendchannel.wait()
```

6.2.3.3.3.2 Escenario: Señal pulsada (sinusoidal)

Incorporar una señal pulsada en el diseño soluciona la posible situación de un resultado erróneo donde visualmente parece correcto. Esto se origina debido a que el desfase originado se repite en el tiempo con un valor proporcional a "x" períodos completos a la señal. De tal manera que, aunque las señales se representan solapadas asemejando la coherencia, el resultado no es válido.

La manera de crear este escenario es muy similar al anterior. En este caso, el contenido del *buffer* que es enviado se divide a la mitad, donde la primera parte está compuesta por ceros mientras que la segunda viene introducida la señal sinusoidal.

Sabiendo que todas las señales se transmiten a la vez, la visualización de este escenario por medio de un osciloscopio surge a partir de su captura, incorporando el *trigger* a cierta amplitud de la señal.

```
n1=8*1024
n2=8*1024
def DMA_Sine_pulsada(freq):
        a      = 1
        f      = freq*1000000
        fs     = 9.984*1000000
        phi    = 0
        signal_gen = sine_gen(a, f, fs, phi, n2)
        array_fp = to_fp(signal_gen[0])
        for i in range(n1):
             in_buffer[i] = 0
        for i in range(n2):
             in_buffer[i] = array_fp[i]
```

6.2.3.3.4 Configuración del bloque de Control de flujo

La función de control de flujo se realiza a través de la introducción del valor "1" para la reanudación del flujo o "0" para su detención. Se utiliza principalmente para incorporar una nueva configuración al diseño cuando se quiere mantener la *multisincronización entre tiles.*

```
disassert = 0x0
mask = 0x1
gpio_dac_ctrl = overlay.ip_dict["stage_DAC/axi_gpio_dac_ctrl"]
dac_ctrl = AxiGPIO(gpio_dac_ctrl).channel1
dac_ctrl.write(disassert,mask)
```

6.2.3.4 Implementación de módulos para recepción

De igual manera a la etapa de transmisión es necesario indicar el *path* para localizar dichos módulos respecto al *Overlay*.

6.2.3.4.1 *Configuración de la DMA* (Modo "escritura")

El formato para esta modalidad es similar a la descrita en la etapa de transmisión, incorporando en este caso la propiedad que realiza la escritura.

Las dos funciones son:

- *rcvdchannel.transfer(X)*: Sirve para recibir en el PS el buffer enviado por parte del PL, siendo "X" el buffer de salida.
- *rcvdchannel.wait()*: Tiene la misma función que en el caso anterior, realiza la espera necesaria para que se complete la acción.

En este apartado, también es necesario operar con el módulo "tlast_gen" para que la DMA realice su función. Para ello, se introduce el valor de muestras que contiene el buffer de salida a través de una GPIO.

```
gpio_tlast = overlay.ip_dict["stage_ADC/axi_gpioRX_tlast"]
tlast = AxiGPIO(gpio_tlast).channel1
tlast.write(num, 0xFFFFFFFF)
```

7 Resultados

En este apartado se muestran los resultados obtenidos del diseño, una vez realizada su implementación y configuración mediante software.

Para la obtención de resultados se ha necesitado utilizar material de laboratorio, específicamente para el campo de las radiofrecuencias. Estas herramientas de trabajo son:

- **Osciloscopio**: Para realizar la visualización de señales cuya frecuencia no alcance 1.25 GSPS (sin aliasing), conociendo que su tasa de muestreo se encuentra a 2.5 GSPS.
- **Analizador de espectros**: Para visualizar las características de la señal y conocer la frecuencia a la que oscila y su amplitud.
- **Generador de señales**: Para la obtención de una señal a cualquier frecuencia, siendo necesario para la realización de pruebas en la etapa de recepción.

En ausencia del generador de señales, otra posibilidad es la realización de un *loopback* del DAC al ADC. El procedimiento sería generar una señal en la etapa de transmisión, modularla a una frecuencia portadora y mandar dicha señal del DAC al ADC. De esta manera, en la etapa de recepción se puede obtener una señal con ciertas características generada por el diseñador.

Respecto a los problemas encontrados, el principal problema es que este tipo de placas de evaluación no ofrece soporte para la integración de aplicaciones. Como su nombre indica, solo realiza tareas de evaluación.

La ZCU111 trae consigo una tarjeta para las conexiones de DACs y ADCs a través de puertos SMA (incorporando *balun* para la conversión de puertos diferenciales a modo común). Esta placa realiza la conversión de la información obtenida por los transceptores GTY. Sin embargo, no están caracterizadas, es decir, existe diferencia de longitud de pista entre los diferentes puertos. Además de que operan a distinta banda de frecuencia, impidiendo alcanzar una coherencia precisa entre canales.

La placa de conexiones de la ZCU111 es la “XM500 RFMC Balun”, se muestra en la Figura 7-1.

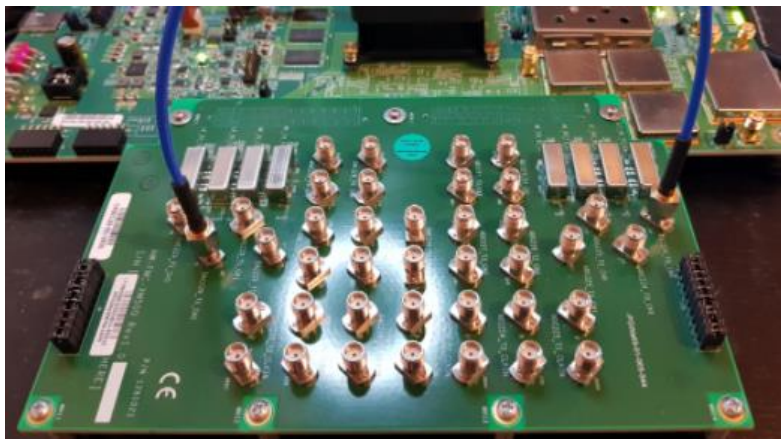

Figura 7-1: Tarjeta “XM500 RFMC Balun” [27]

Para resolver este problema, se ha podido trabajar en la etapa de transmisión mediante una placa adaptada de la empresa SENER (Figura 7-2) con la que se obtiene un desfase casi nulo para los 8 canales.

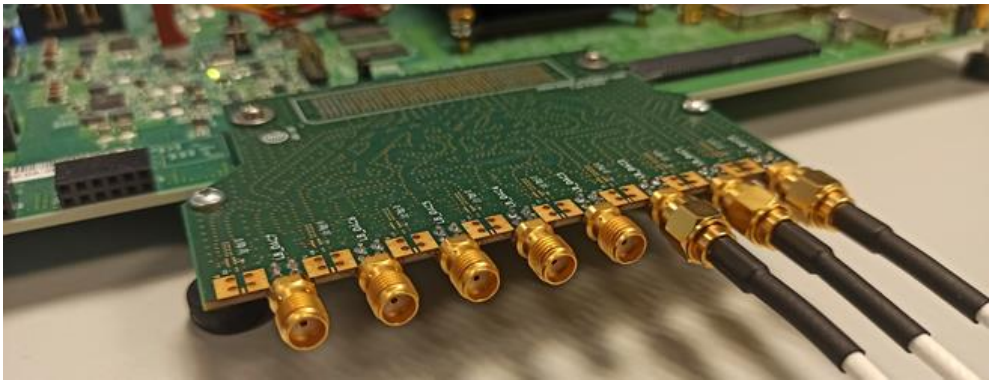

Figura 7-2: Tarjeta adaptada (SENER)

Respecto a la etapa de recepción, no se ha podido trabajar con una placa acondicionada (no existe una placa dual para recepción en modo común para los ADCs). En este caso, por la tecnología utilizada actualmente, la obtención de resultados no es nada concluyente. Aun así, ya que los componentes del mismo *tile* comparten la misma estructura, se ha realizado un análisis de desfase para ADCs del *tile 0*, permitiendo introducir una etapa de calibrado para poder compensar cierta diferencia de fase.

Cabe recordar que la estructura de *tiles* de la ZCU111 es la siguiente:

- **DAC**: *Tile 0* (DAC0, DAC1, DAC2 y DAC3) y *Tile 1 (*DAC4, DAC5, DAC6 y DAC7*)*
- **ADC**: *Tile 0 (*ADC0 y ADC1*), Tile 1 (*ADC2 y ADC3*), Tile 2 (*ADC4 y ADC5*)* y *Tile 3 (*ADC6 y ADC7*).*

Aquellos componentes que se encuentran en el mismo *tile* comparten el mismo reloj, por tanto, existe sincronización entre ellos sin necesidad de aplicar la funcionalidad *runMTS()*.

A continuación, se muestran los resultados de ambas etapas.

7.1 Resultados de la etapa de transmisión

En este apartado se muestran los resultados para la etapa de transmisión. Es importante conocer que a la salida de la señal no se ha introducido un filtro paso bajo analógico a TM/2, siendo TM la tasa de muestro elegida. Esto implica el deterioro de la señal modulada por la incorporación de espurios (debido al solapamiento de réplicas).

Para esta etapa de transmisión, la obtención de resultados tiene dos fases:

- La primera fase consiste en validar la “coherencia entre canales” (*MTS*). Para ello, se va a utilizar el osciloscopio. Con esta herramienta se puede observar de una manera sencilla el desfase entre canales tanto para mismo *tile* como distinto *tile*.
  Para conocer el valor del desfase, se utiliza la opción de medida “delay” que trae consigo.
- La segunda parte se trata de comprobar los resultados a través de un analizador de espectros, con esto se comprueba el estado de la señal en todo el ancho de banda

efectivo.

7.1.1 Pruebas con osciloscopio

7.1.1.1 Escenario: Señal sinusoidal

A continuación, se muestran los resultados del desfase obtenido para los diferentes casos. Todas las señales operan en la misma frecuencia (2.5 MHz) y, para realizar el desfase, se ha impuesto como DAC de referencia el primero del *tile* 0 (DAC0).

Los valores de desfase obtenidos a partir del osciloscopio no llegan a ser exactos ya que representan el muestreo reconstruido. Dicha diferencia se acentúa con el aumento de frecuencia tras la modulación de la señal.

- **Pruebas sin *MTS* (SIN *multisincronización entre tiles*)**

Señales en banda base (Mezclador en *bypass*), 2.5 MHz

Como se comentaba anteriormente, cuando se trabaja a baja frecuencia se complica la obtención de resultados, ya que las medidas no son tan precisas. Se obtienen valores que oscilan en un rango de ±2.5 ns.

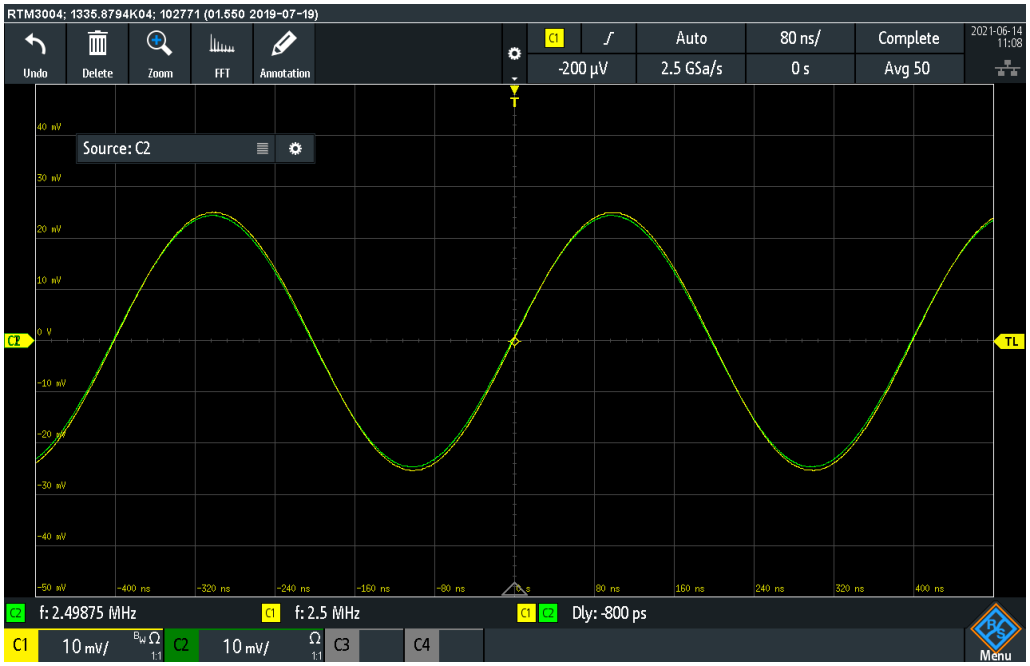

Figura 7-3: Desfase DAC0-DAC1 (Banda base) - 2.5 MHz

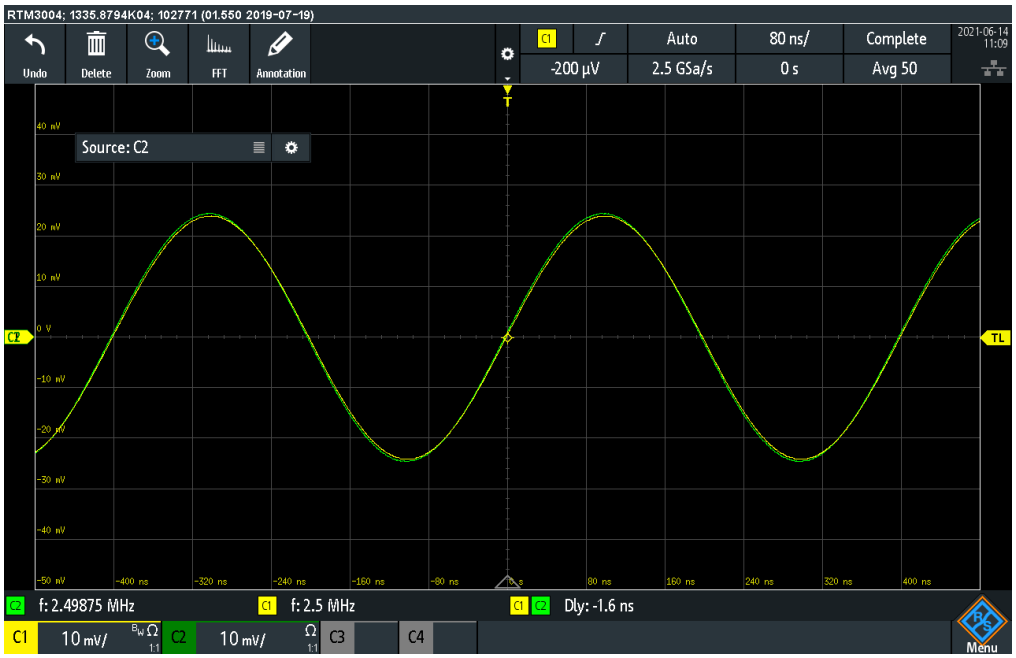

Figura 7-4: Desfase DAC0-DAC2 (Banda base) - 2.5 MHz

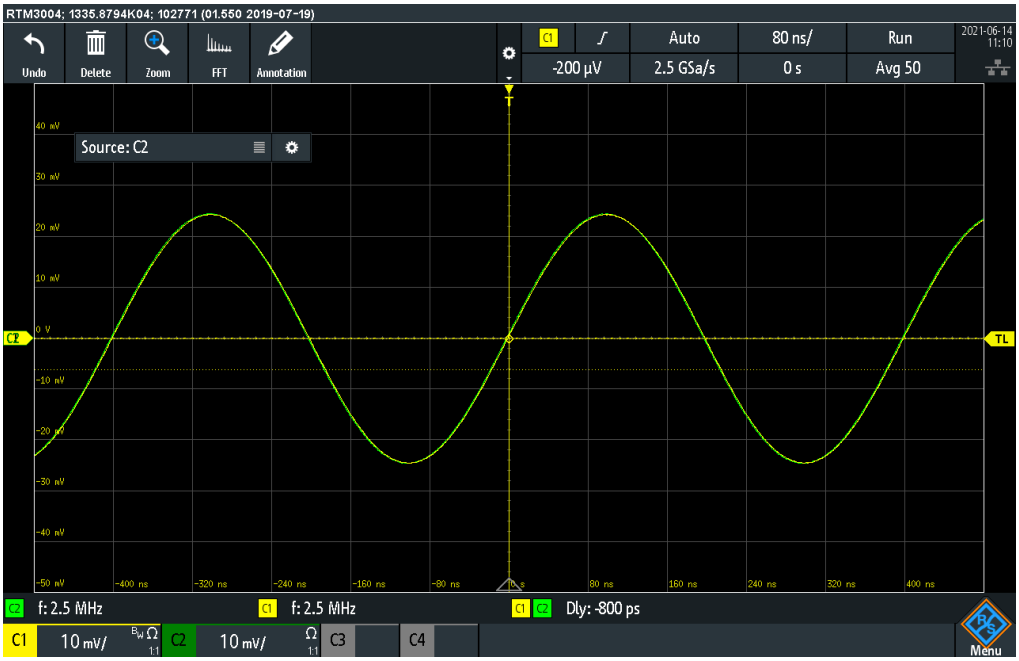

Figura 7-5: Desfase DAC0-DAC3 (Banda base) - 2.5 MHz

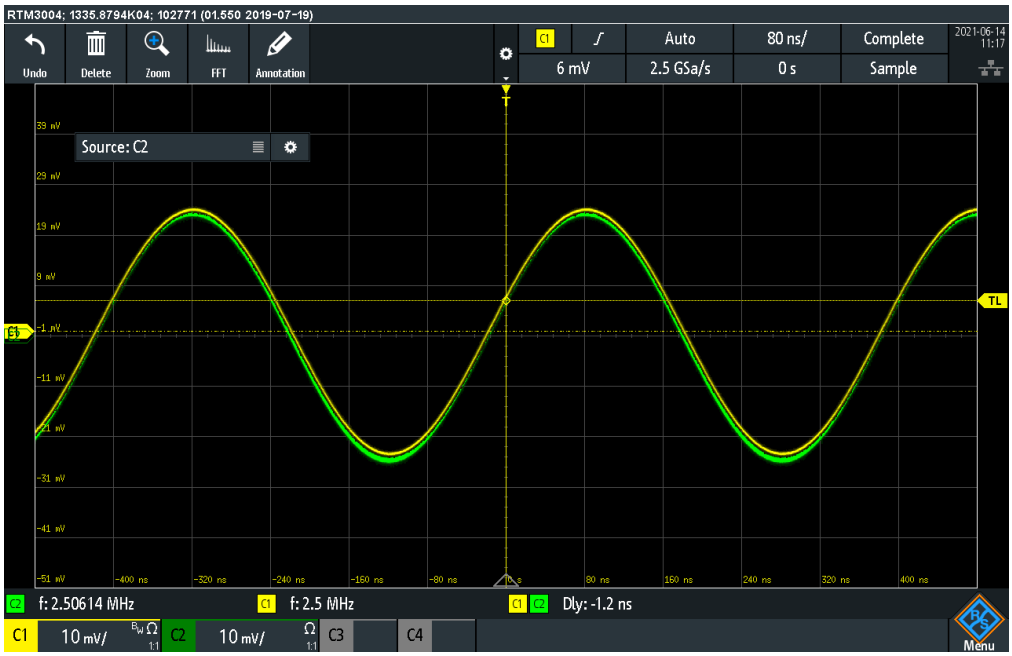

Figura 7-6: Desfase DAC0-DAC4 (Banda base) - 2.5 MHz

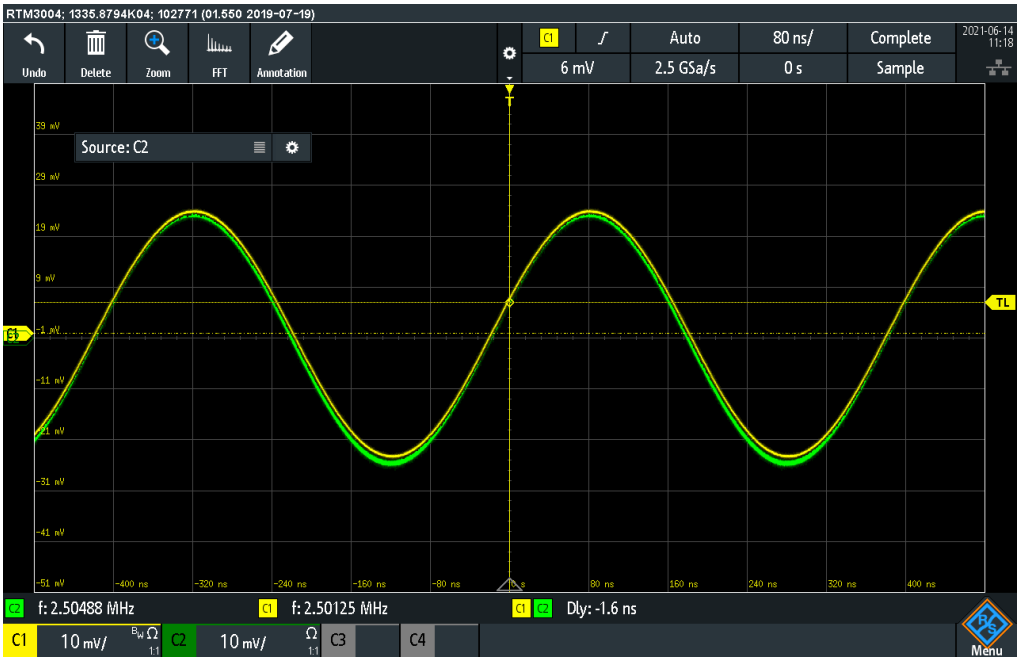

Figura 7-7: Desfase DAC0-DAC5 (Banda base) - 2.5 MHz

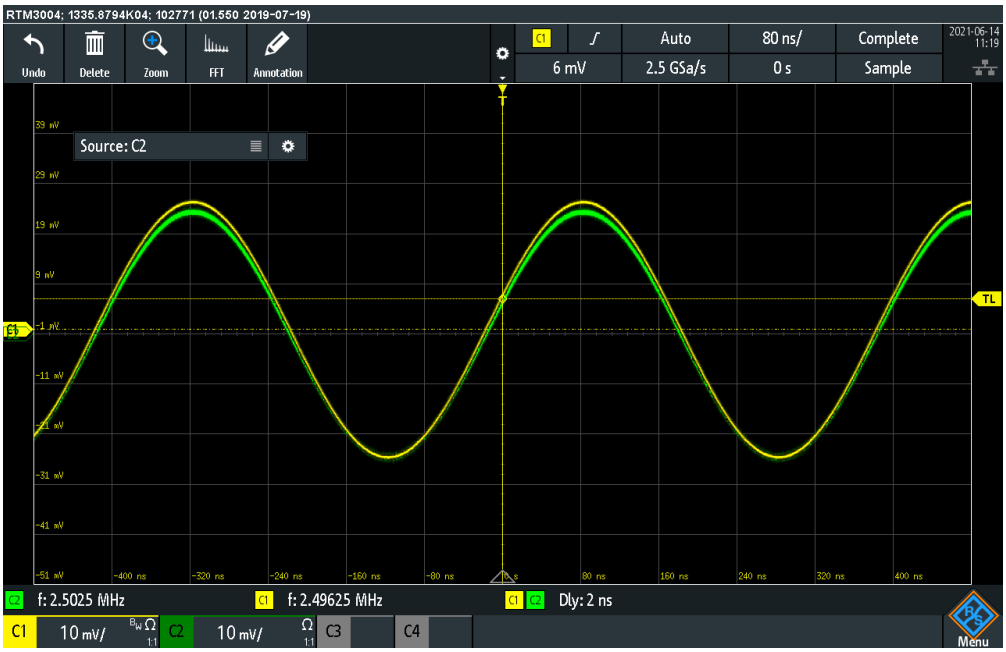

Figura 7-8: Desfase DAC0-DAC6 (Banda base) - 2.5 MHz

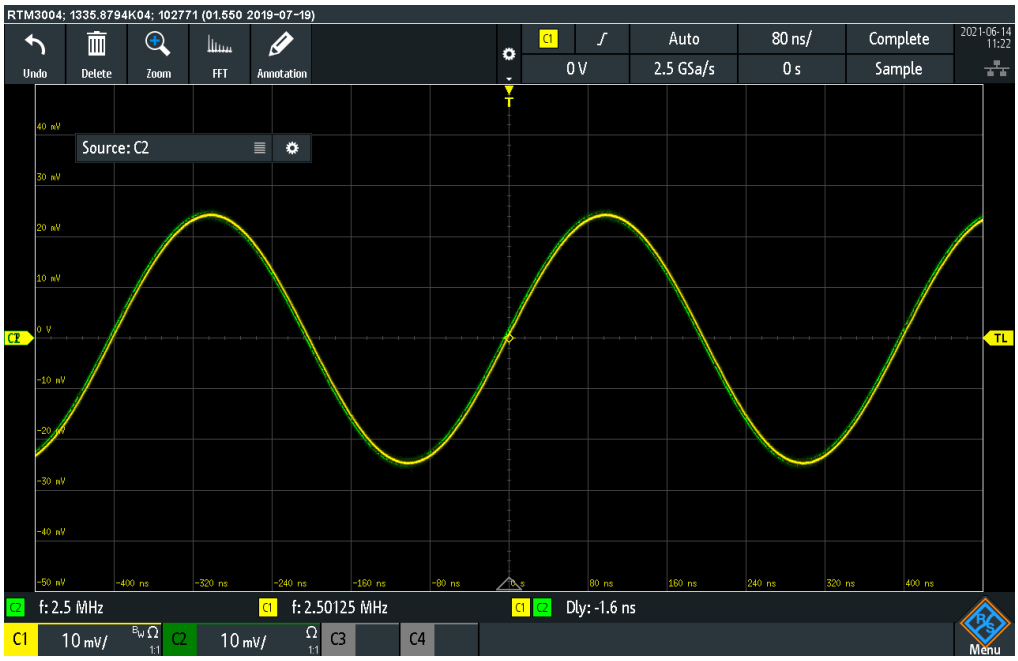

Figura 7-9: Desfase DAC0-DAC7 (Banda base) - 2.5 MHz

Los resultados obtenidos del desfase entre canales para una señal sin modular (en banda base) se muestran en la Tabla 4.

DAC0						
DAC1	DAC2	DAC3	DAC4	DAC5	DAC6	DAC7
-800 ps	-1.6 ns	-800 ps	-1.2 ns	-1.6 ns	2 ns	-1.6 ns

Tabla 4: Desfase DACs (señal en "banda base")

Señales moduladas a 500 MHz con EVNT_SRC_INMEDIATE

En este apartado se muestran los resultados de las señales una vez configurado el mezclador para una frecuencia portadora de 497.5 MHz, obteniendo una señal modulada a 500 MHz.

La característica que tiene EVNT_SRC_INMEDIATE es que no existe evento común, es decir, la modulación se hace instantánea. Por ello, no existe sincronización para ningún caso.

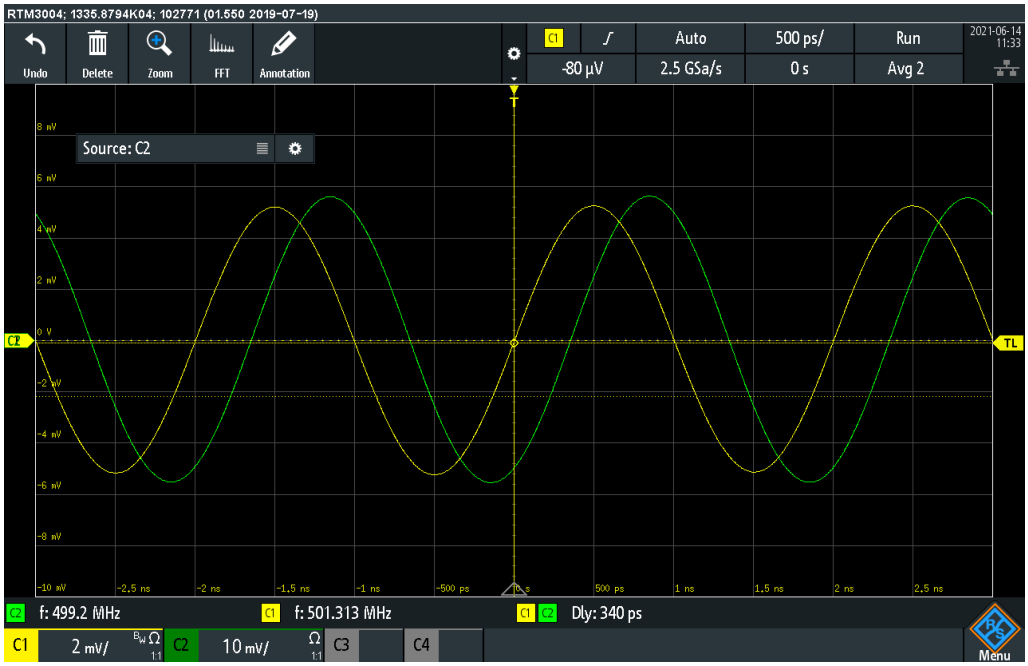

Figura 7-10: Desfase DAC0-DAC1 (INMEDIATE) - 500 MHz

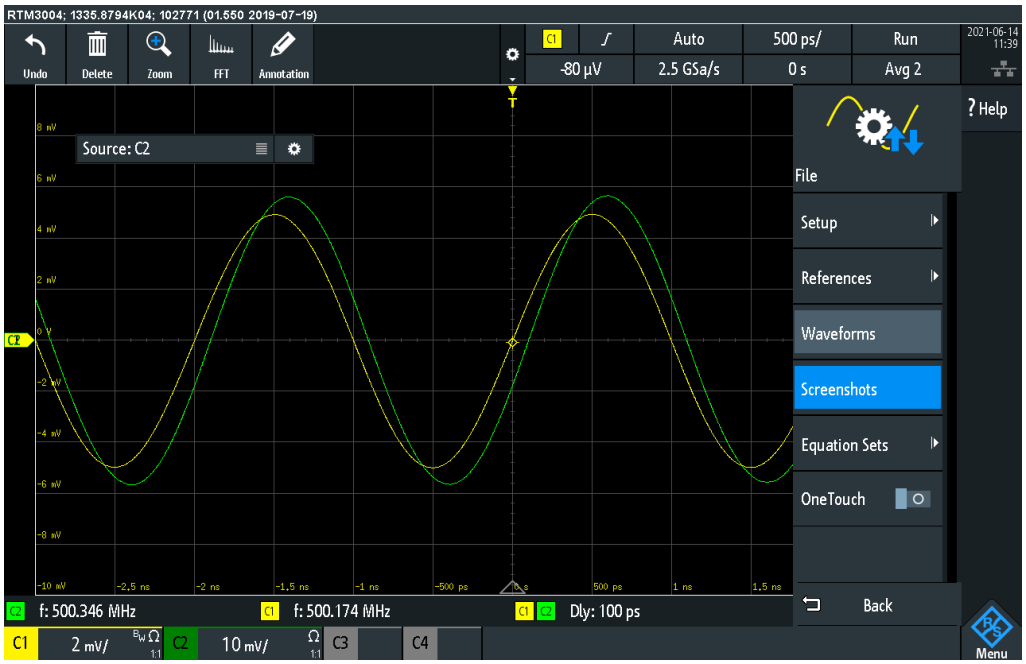

Figura 7-11: Desfase DAC0-DAC2 (INMEDIATE) - 500 MHz

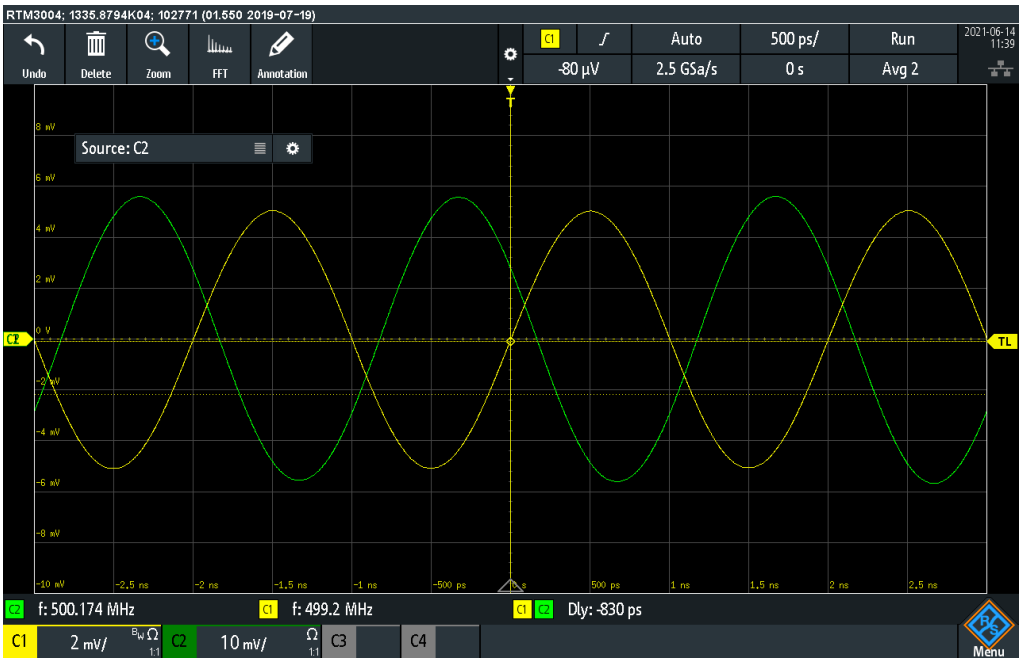

Figura 7-12: Desfase DAC0-DAC3 (INMEDIATE) - 500 MHz

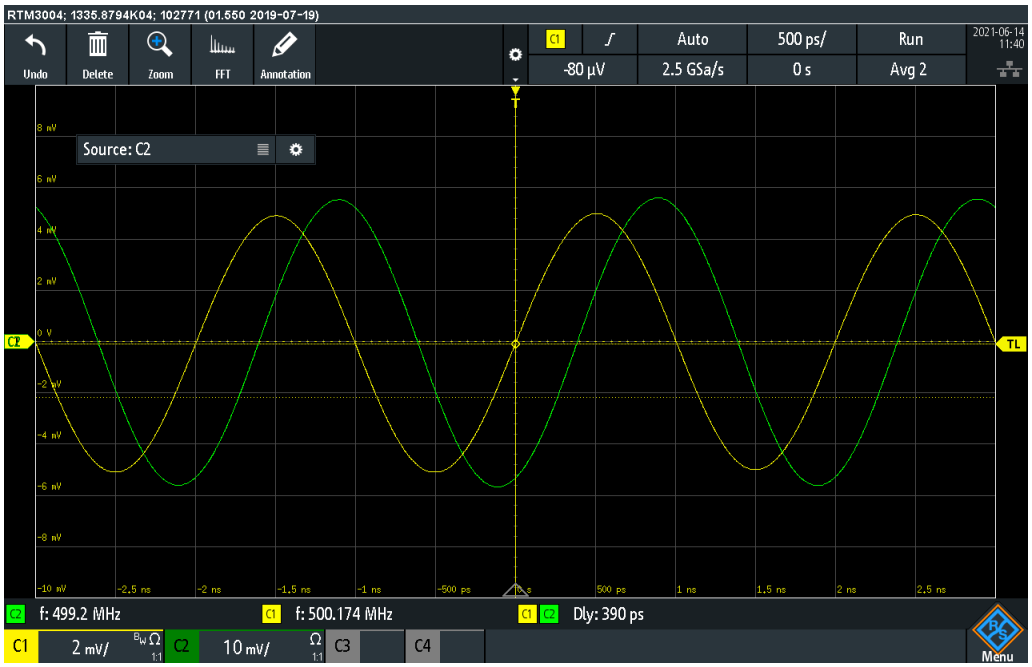

Figura 7-13: Desfase DAC0-DAC4 (INMEDIATE) - 500 MHz

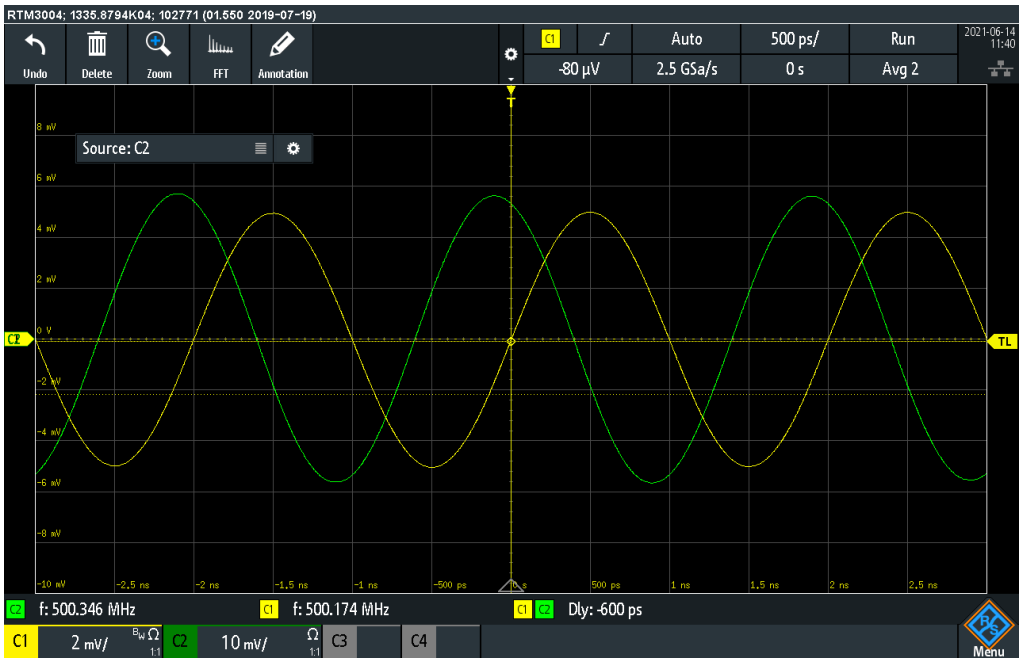

Figura 7-14: Desfase DAC0-DAC5 (INMEDIATE) - 500 MHz

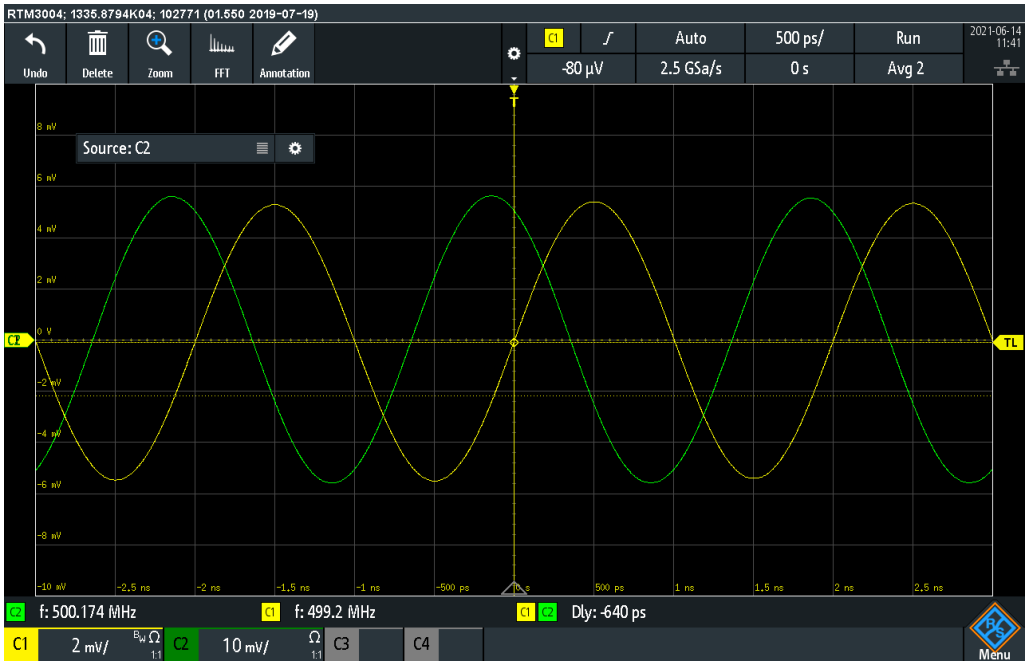

Figura 7-15: Desfase DAC0-DAC6 (INMEDIATE) - 500 MHz

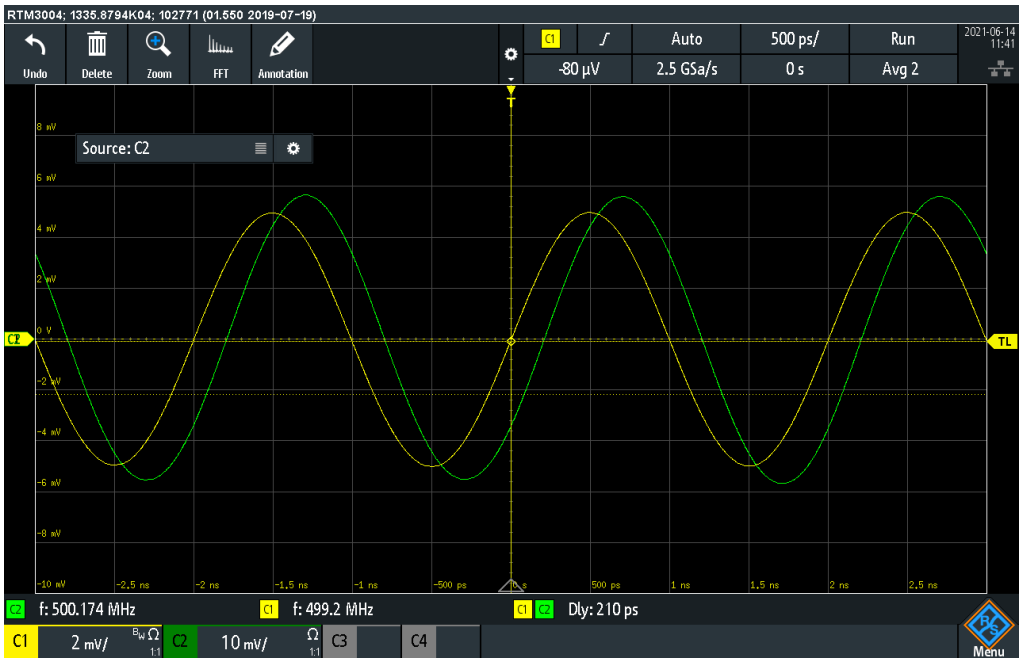

Figura 7-16: Desfase DAC0-DAC7 (INMEDIATE) - 500 MHz

Los resultados obtenidos del desfase entre canales para una señal modulada con tipo de evento (EVNT_SRC_INMEDIATE) se muestra en la Tabla 5.

DAC0						
DAC1	DAC2	DAC3	DAC4	DAC5	DAC6	DAC7
340 ps	100 ps	-830 ps	390 ps	-600 ps	-640 ps	210 ps

Tabla 5: Desfase DACs (señal modulada con evento "INMEDIATE")

Señales moduladas a 500 MHz con EVNT_SRC_TILE

Este apartado sigue la misma línea que en el caso anterior, se obtiene una señal modulada a 500 MHz. Sin embargo, la configuración introducida en el mezclador es EVNT_SRC_TILE, la cual realiza la modulación teniendo en cuenta el reloj común para aquellos componentes del mismo *tile*, obteniendo así coherencia entre esos canales.

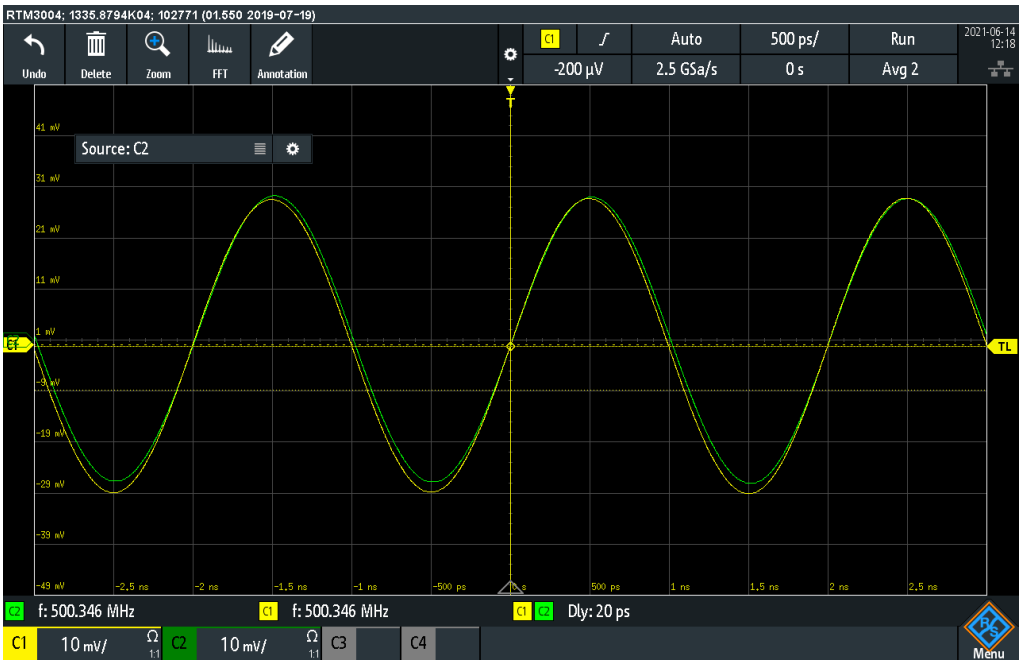

Figura 7-17: Desfase DAC0-DAC1 (TILE) - 500 MHz

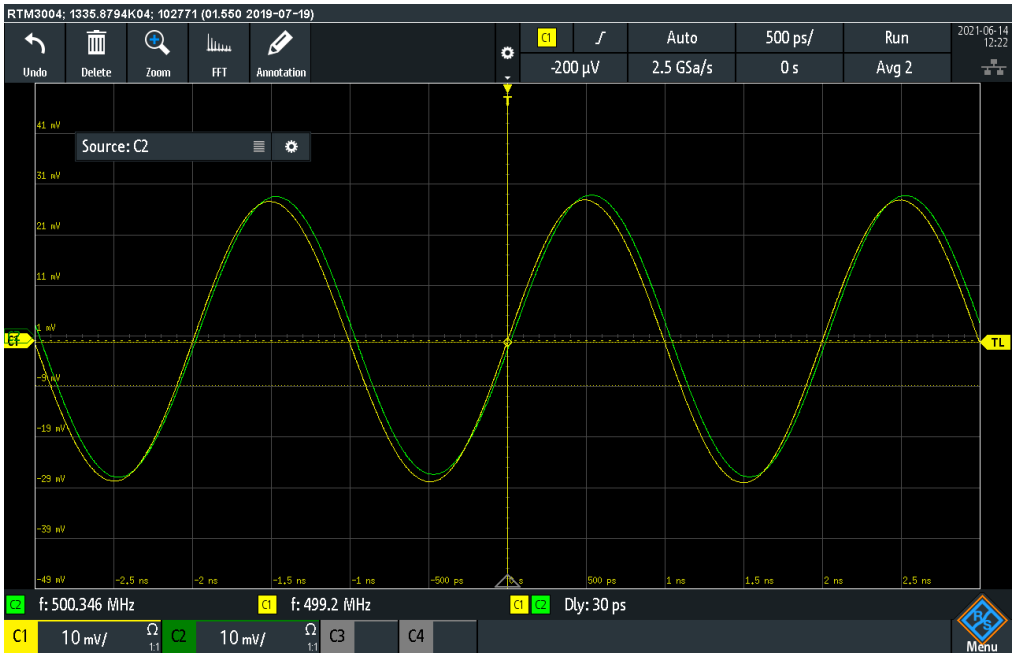

Figura 7-18: Desfase DAC0-DAC2 (TILE) - 500 MHz

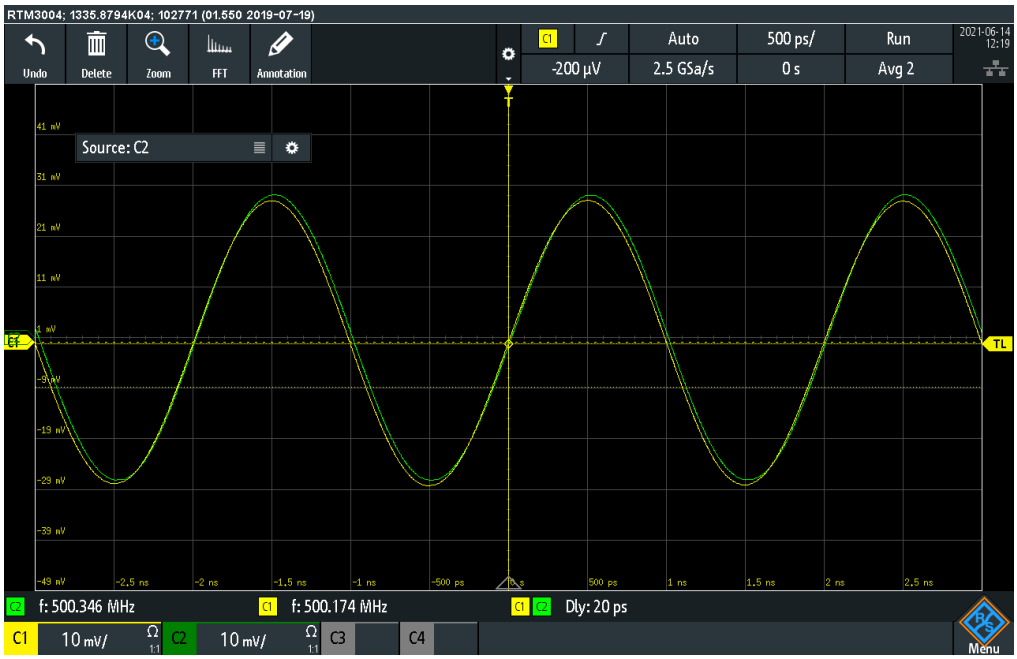

Figura 7-19: Desfase DAC0-DAC3 (TILE) - 500 MHz

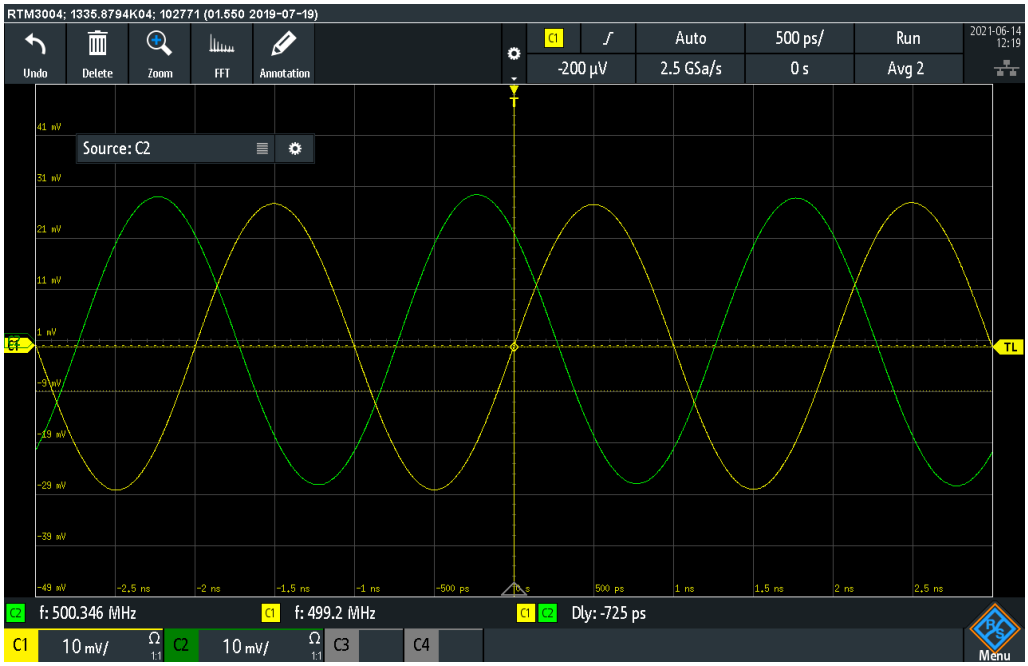

Figura 7-20: Desfase DAC0-DAC4 (TILE) - 500 MHz

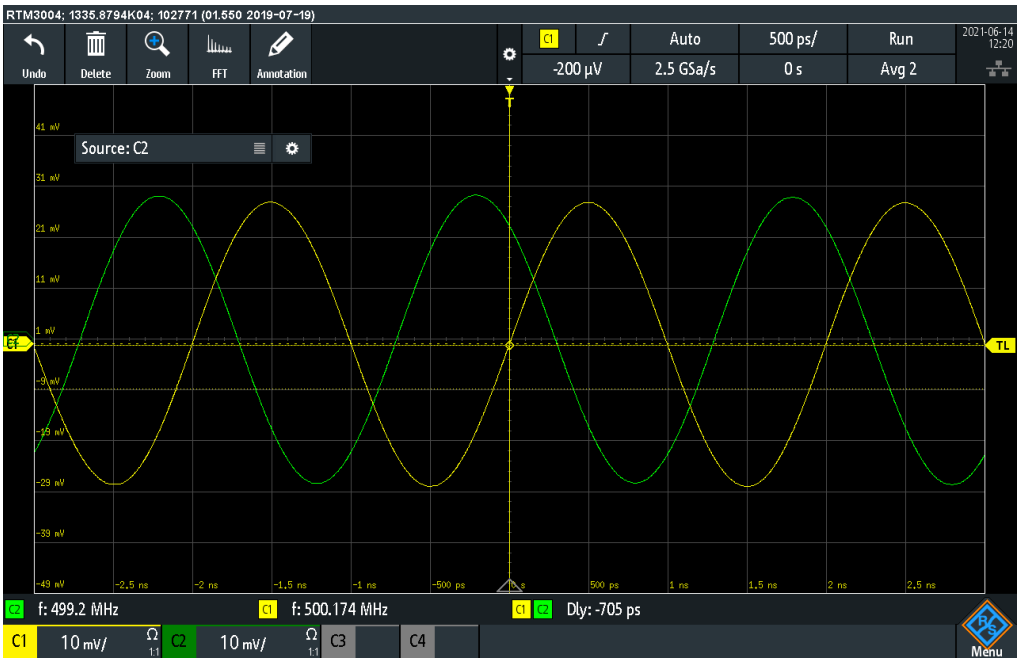

Figura 7-21: Desfase DAC0-DAC5 (TILE) - 500 MHz

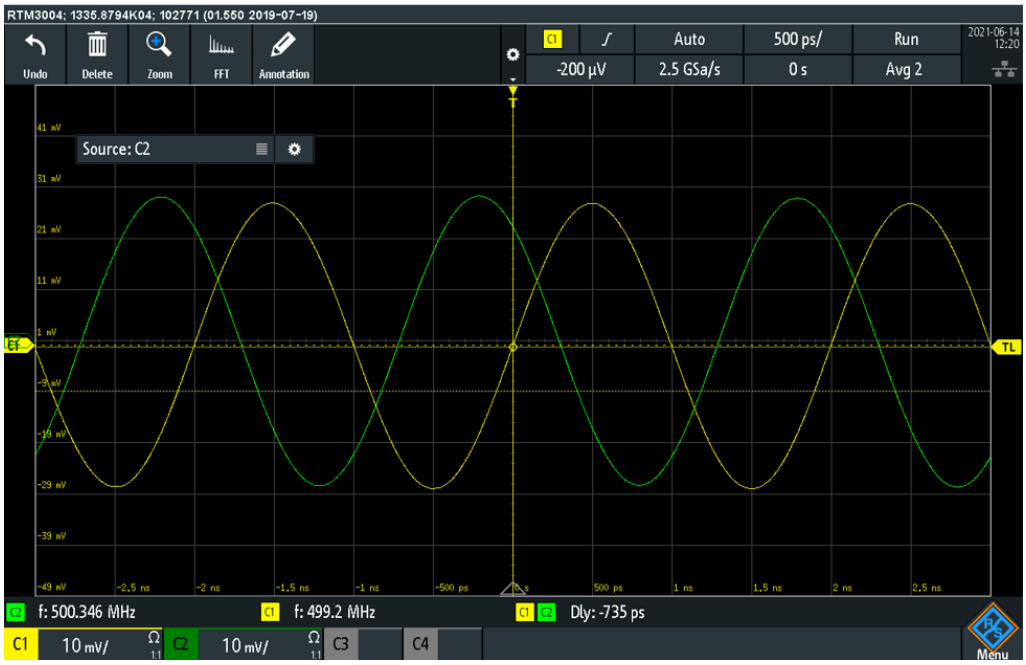

Figura 7-22: Desfase DAC0-DAC6 (TILE) - 500 MHz

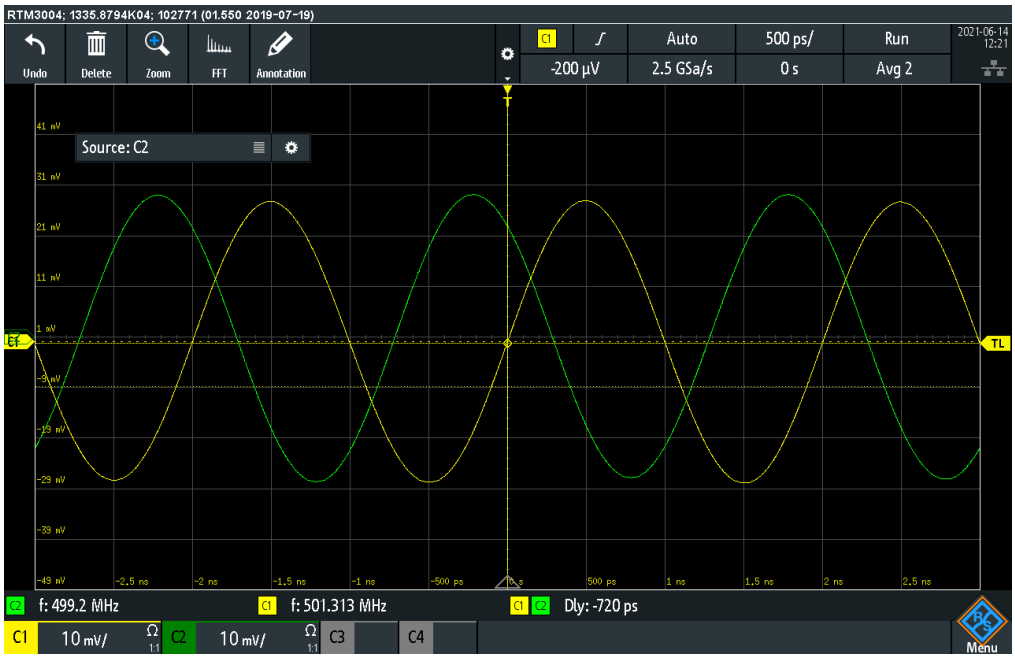

Figura 7-23: Desfase DAC0-DAC7 (TILE) - 500 MHz

Los resultados obtenidos del desfase entre canales para una señal modulada con tipo de evento (EVNT_SRC_TILE) se muestra en la Tabla 6.

DAC0						
DAC1	DAC2	DAC3	DAC4	DAC5	DAC6	DAC7
20 ps	30 ps	20 ps	-725 ps	-705 ps	-735 ps	-720 ps

Tabla 6: Desfase DACs (señal modulada con evento "TILE")

- **Pruebas con *MTS* (*multisincronización entre tiles*)**

En este apartado se introduce como evento común EVNT_SRC_SYSREF, obteniendo la coherencia para todos los canales.

MTS: Señales moduladas a 1 GHz con EVNT_SRC_SYSREF

Para los siguientes resultados se realiza la modulación de la señal a una frecuencia portadora de 997.5 MHz, apareciendo una señal a 1 GHz.

En este apartado se encuentran desfases menores a 30 ps para todos los canales, cumpliendo el requerimiento estipulado.

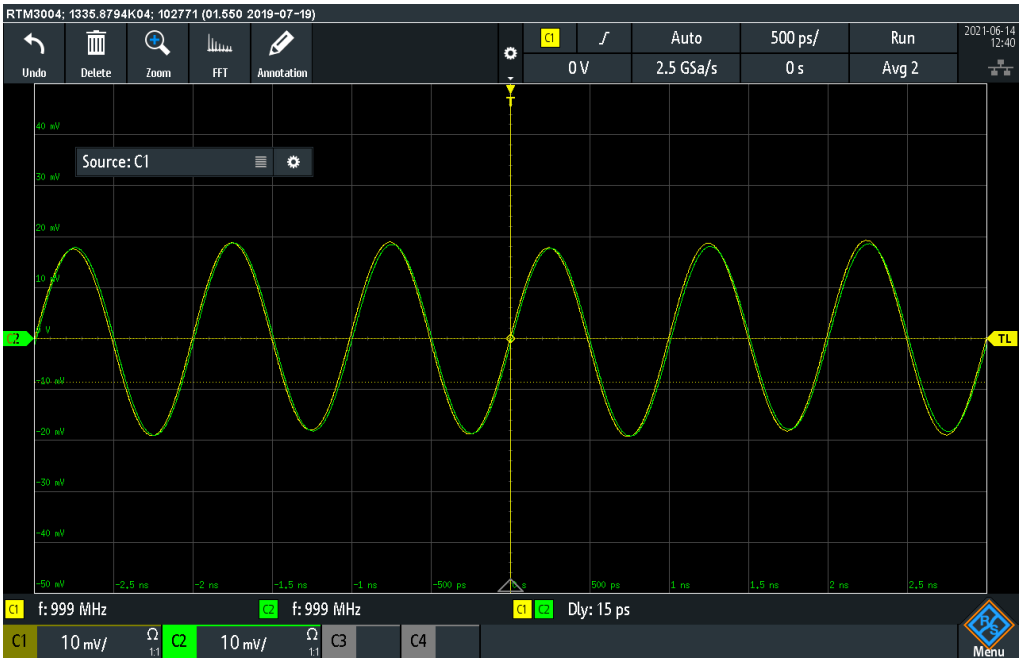

Figura 7-24: Desfase DAC0-DAC1 (SYSREF) - 1 GHz

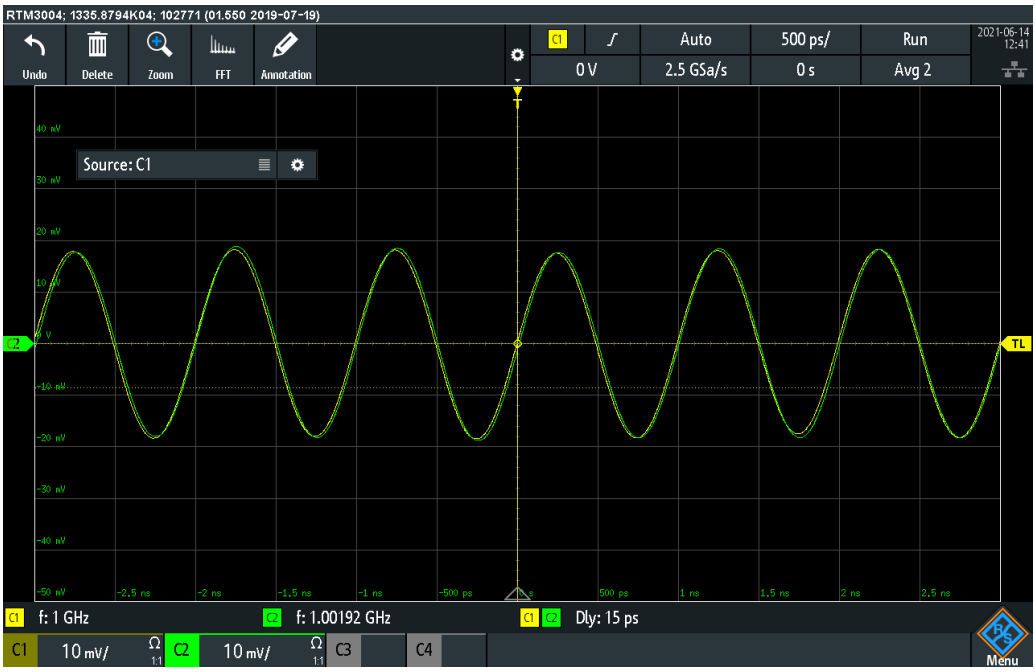

Figura 7-25: Desfase DAC0-DAC2 (SYSREF) - 1 GHz

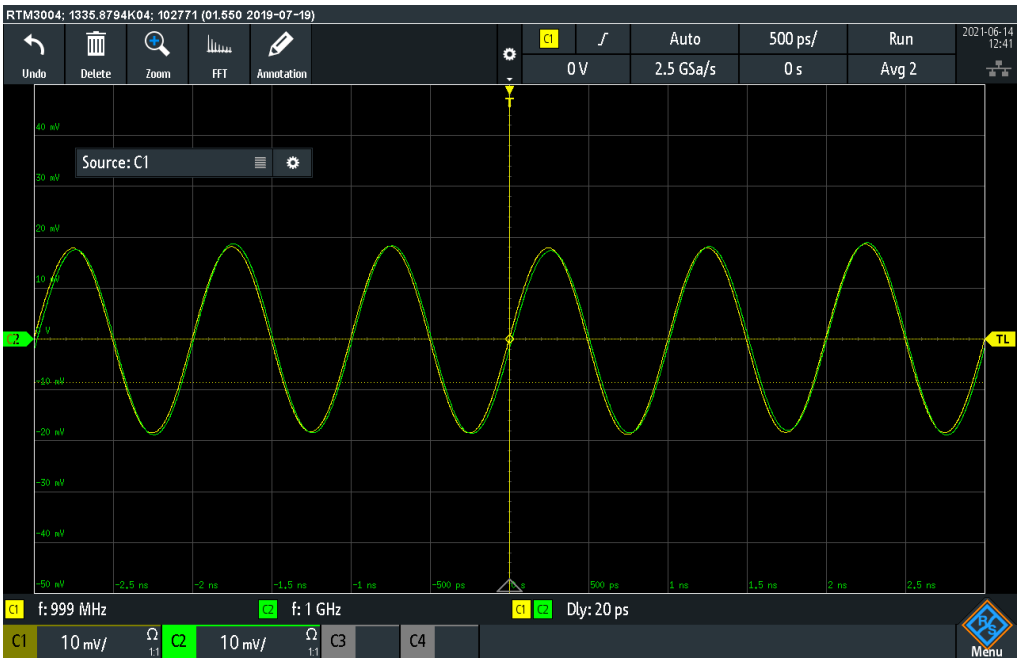

Figura 7-26: Desfase DAC0-DAC3 (SYSREF) - 1 GHz

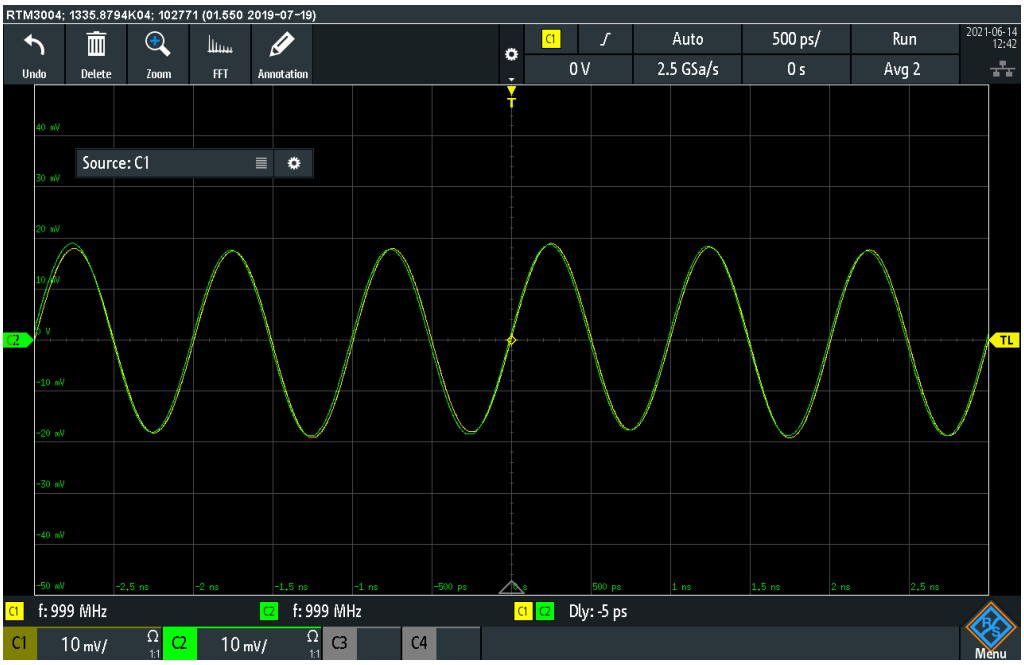

Figura 7-27: Desfase DAC0-DAC4 (SYSREF) - 1 GHz

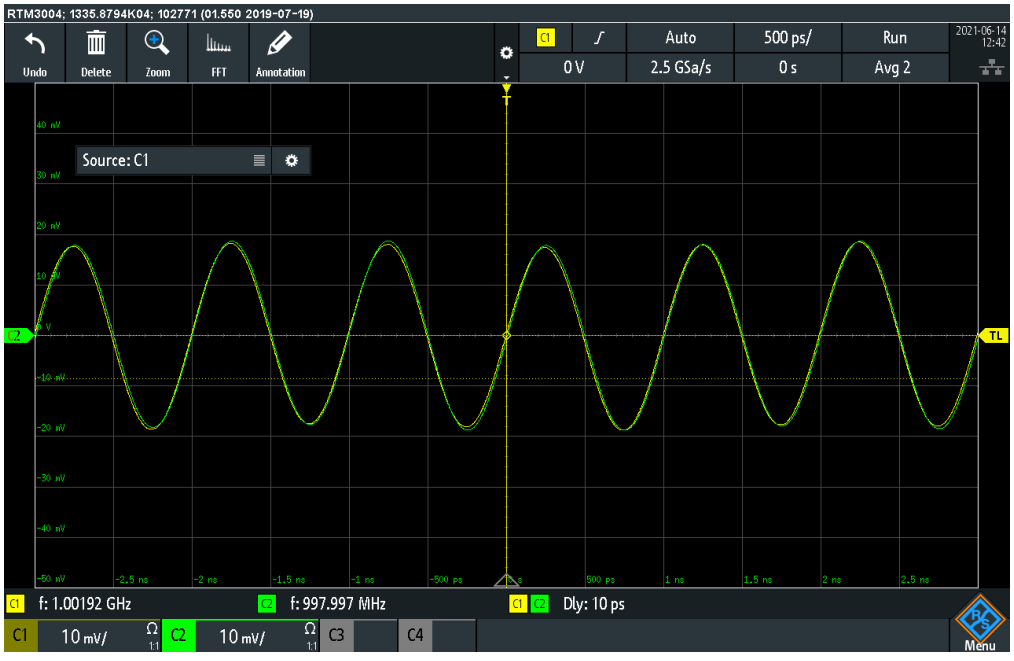

Figura 7-28: Desfase DAC0-DAC5 (SYSREF) - 1 GHz

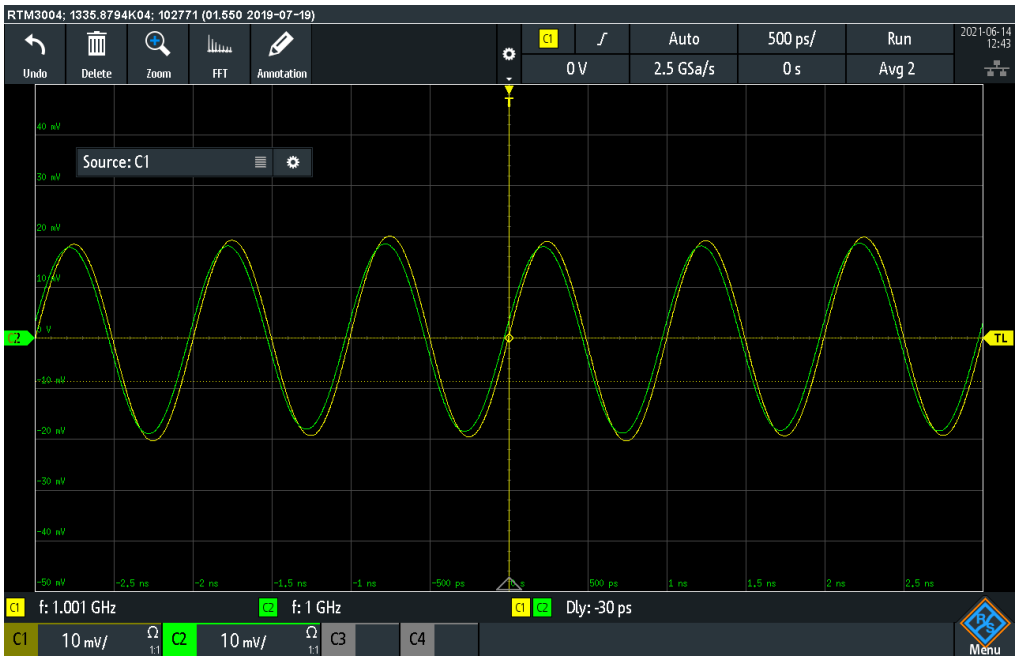

Figura 7-29: Desfase DAC0-DAC6 (SYSREF) - 1 GHz

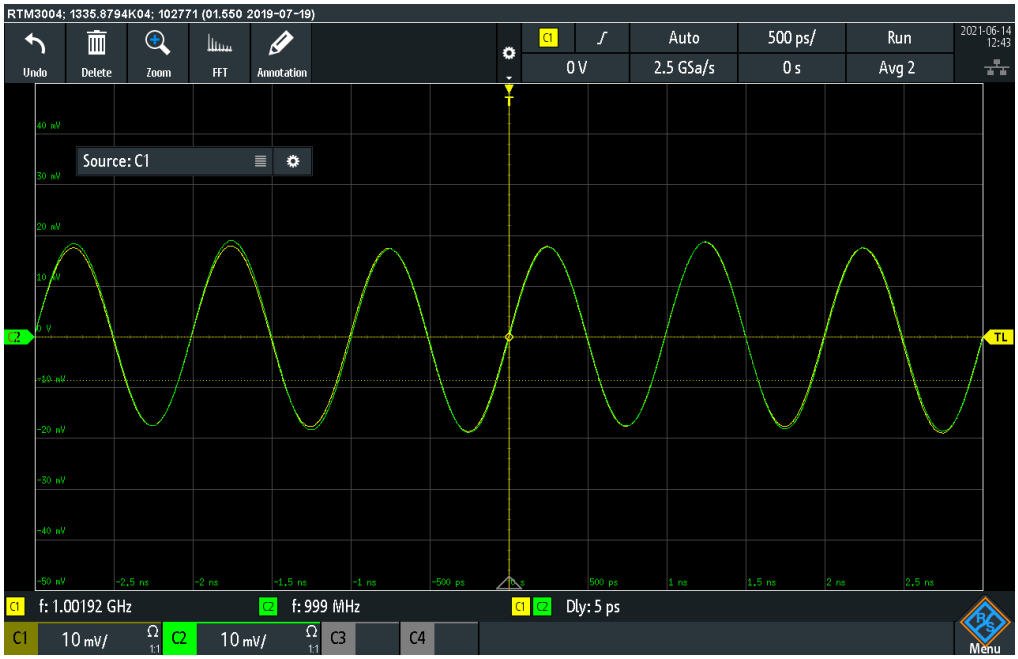

Figura 7-30: Desfase DAC0-DAC7 (SYSREF) – 1 GHz

Los resultados obtenidos del desfase entre canales para una señal modulada con tipo de evento (EVNT_SRC_SYSREF) se muestra en la Tabla 7.

DAC0						
DAC1	DAC2	DAC3	DAC4	DAC5	DAC6	DAC7
15 ps	15 ps	20 ps	-5 ps	10 ps	-30 ps	5 ps

Tabla 7: Desfase DACs (señal modulada con evento “SYSREF”)

Una vez comprobada la coherencia, se lleva a cabo la comprobación de la señal obtenida a lo largo del ancho de banda efectivo.

MTS: Señales moduladas a 1.2 GHz con EVNT_SRC_SYSREF

En este apartado se modula la señal a un valor muy cercano al máximo alcance que se obtiene del osciloscopio (1.25 GSPS sin alising). Se puede comprobar a tiempo real que se deteriora la señal, visualizando la aparición de *jitter*.

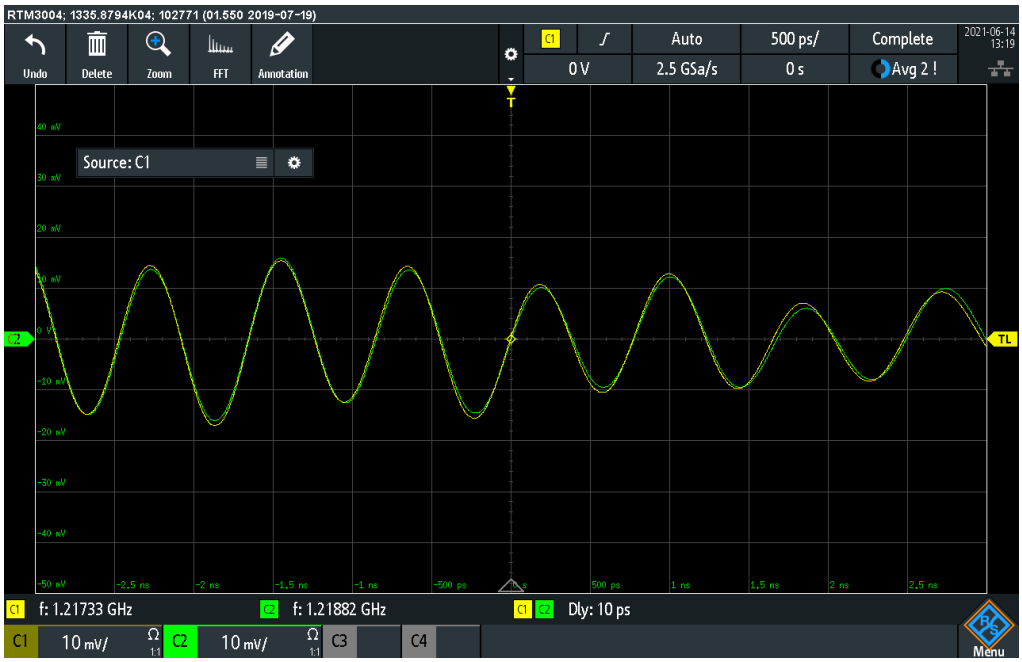

Figura 7-31: Desfase DAC0-DAC1 (SYSREF) – 1.2 GHz

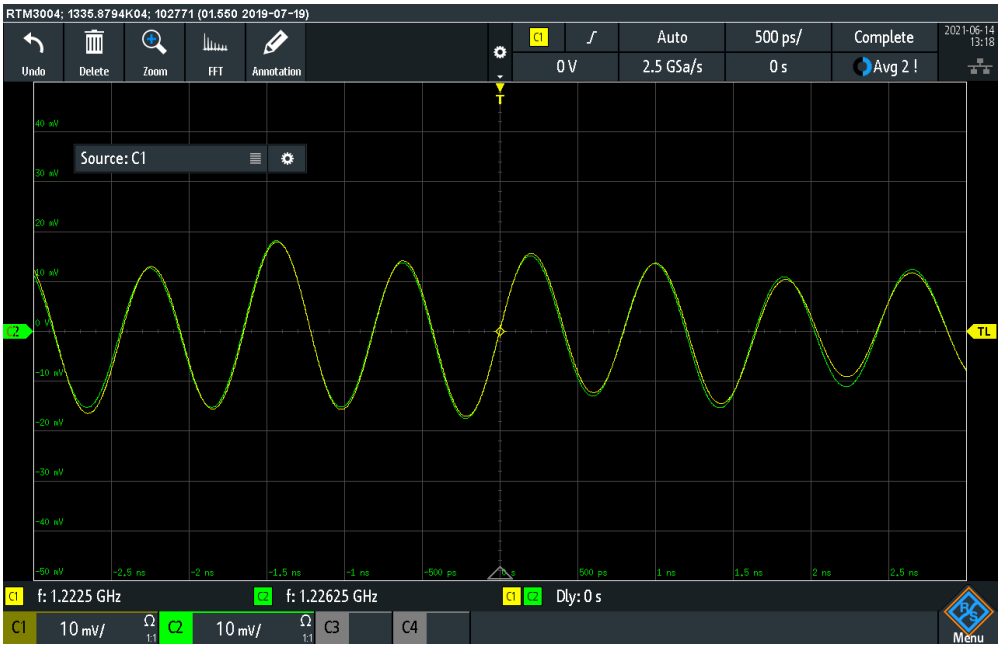

Figura 7-32: Desfase DAC0-DAC4 (SYSREF) – 1.2 GHz

MTS: Señales moduladas a 1.7 GHz con EVNT_SRC_SYSREF

En este punto, la señal observada excede el alcance del osciloscopio. La visualización mostrada en la Figura 7-33 es la combinación de la señal modulada a 1.7 GHz y la réplica de esta (situada a 1.494 GHz). Ya que esta superposición de señales sobrepasa la TM/2 del osciloscopio, el filtro antialiasing que incorpora la herramienta atenúa el resultado, pero no lo suficiente, obteniendo una señal mal muestreada.

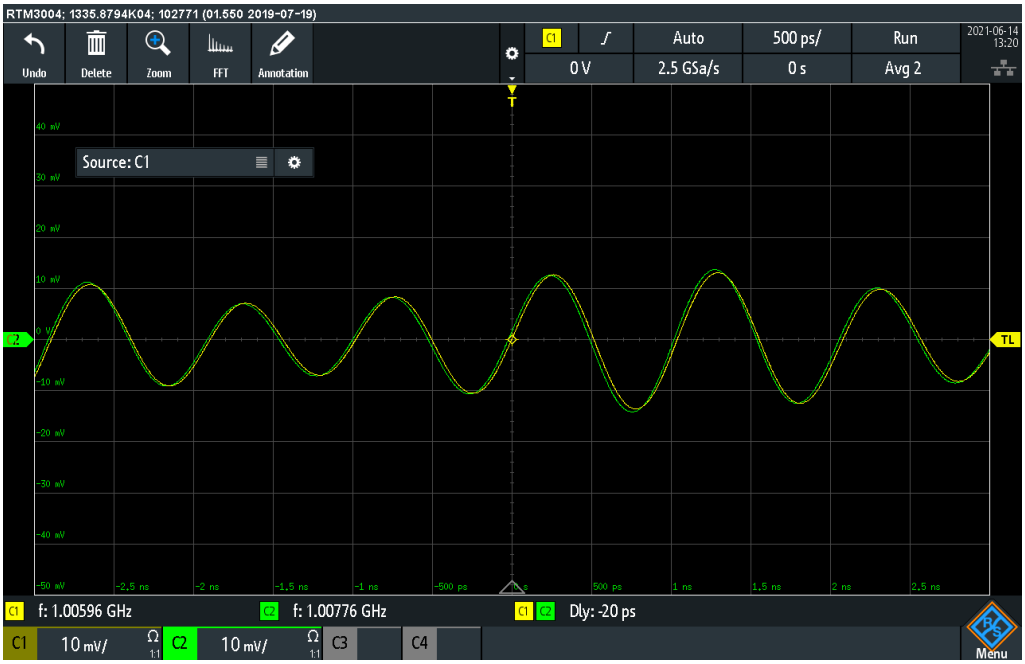

Figura 7-33: Desfase DAC0-DAC1 (SYSREF) – 1.7 GHz

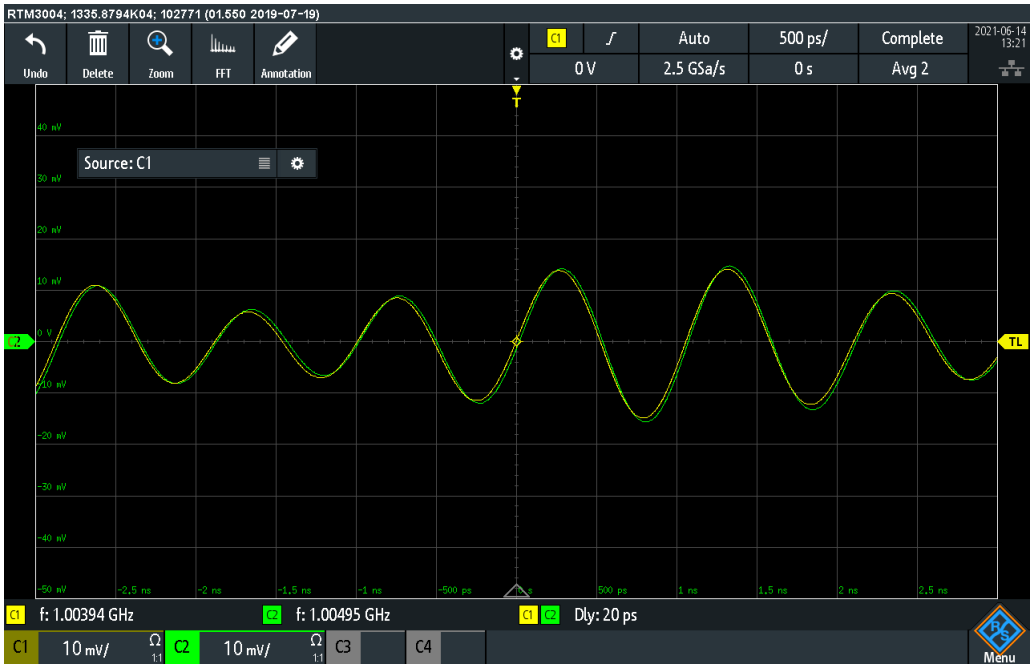

Figura 7-34: Desfase DAC0-DAC4 (SYSREF) – 1.7 GHz

MTS: Señales moduladas a 2.3 GHz con EVNT_SRC_SYSREF

En este último caso, el filtro antialiasing elimina la señal modulada a 2.3 GHz. Sin embargo, se incorpora la réplica a 894 MHz, ya que se sitúa por debajo de la frecuencia de corte.

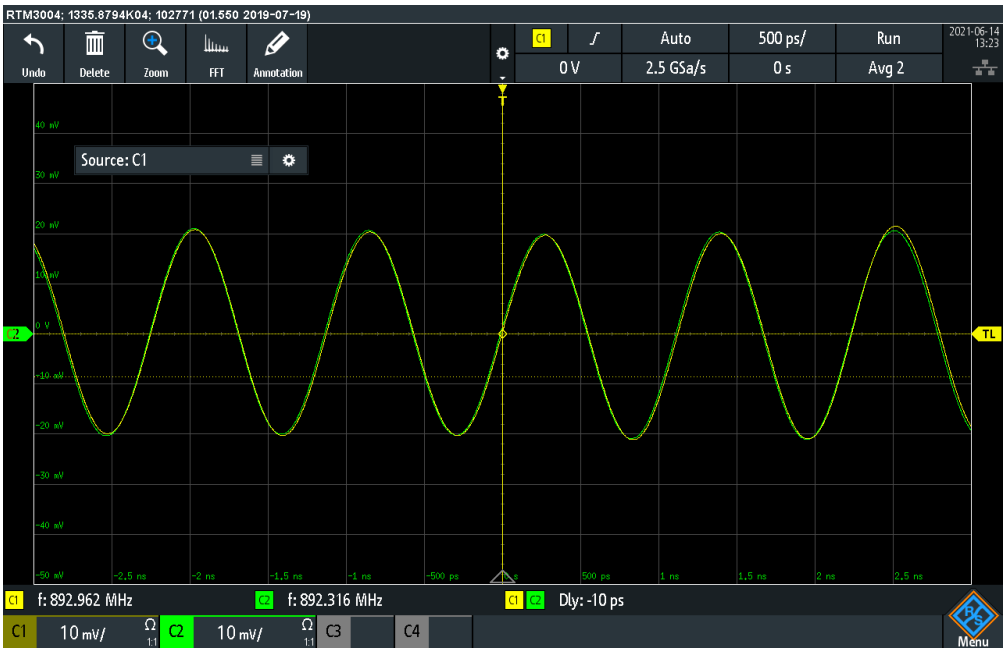

Figura 7-35: Desfase DAC0-DAC1 (SYSREF) – 2.3 GHz

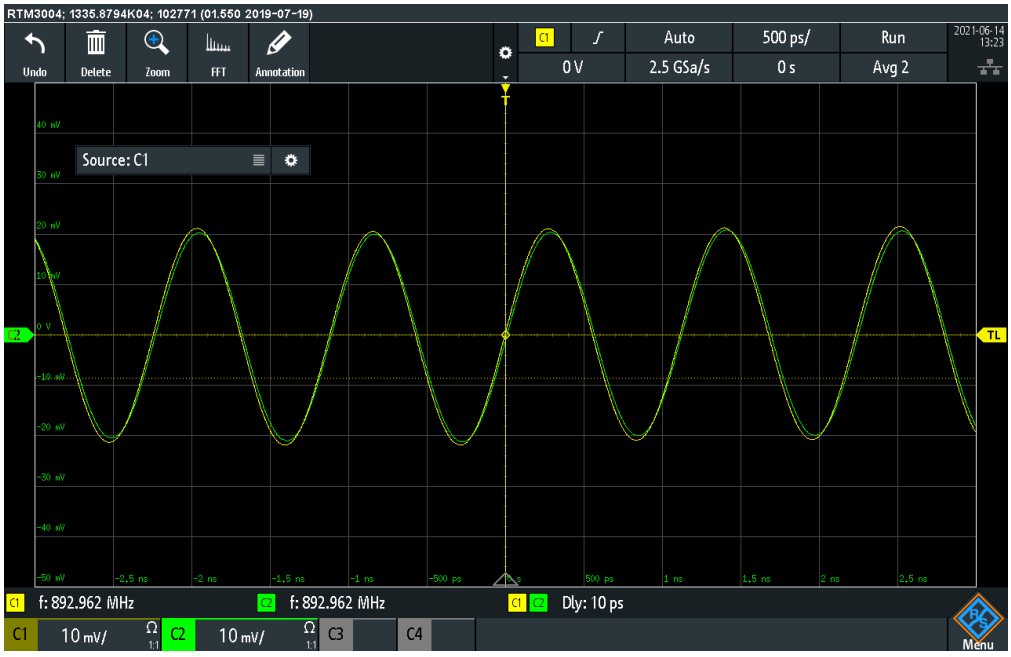

Figura 7-36: Desfase DAC0-DAC4 (SYSREF) – 2.3 GHz

7.1.1.2 Escenario: señal pulsada

- **Pruebas con *MTS* (*multisincronización entre tiles*)**

A continuación, se muestran los resultados obtenidos a partir de introducir una señal pulsada. Esto permite garantizar que la sincronización no es errónea, es decir, no se ha dado la situación en que dichas señales se haya formado un desfase proporcional al período de la señal, asemejando la "coherencia entre canales" cuando de verdad no existe.

En la Figura 7-37 se muestra el desfase encontrado entre el DAC0 y el DAC1. Al ser componentes del mismo *tile*, existe sincronización. Se puede observar que ambas señales se encuentran totalmente solapadas (apenas se pueden diferenciar). También se puede apreciar al comienzo de la señal su parte transitoria.

Figura 7-37: Desfase Señal pulsada DAC0-DAC1 (BB) – 2.5 MHz

En cambio, en la Figura 7-38 se muestra el desfase encontrado entre DAC0-DAC4 (distinto *tile*). Se puede apreciar que existe cierto desfase (muy pequeño) proporcional a lo observado cuando se opera a baja frecuencia (2.5 MHz).

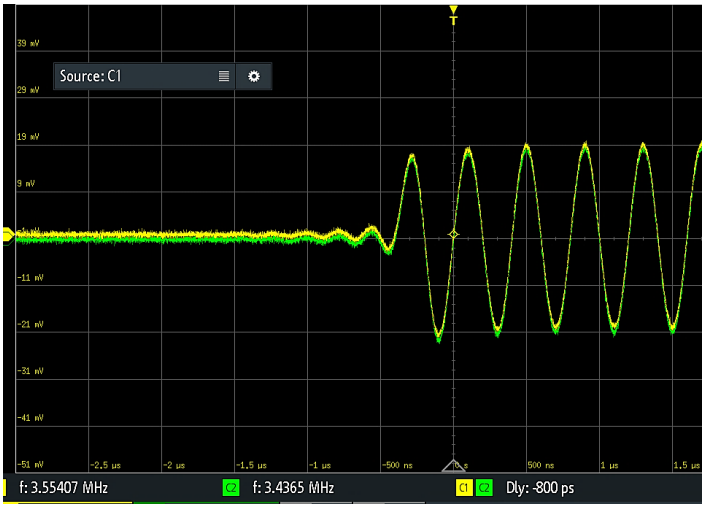

Figura 7-38: Desfase Señal pulsada DAC0-DAC4 (BB) – 2.5 MHz

A continuación, se muestran dichas señales pulsadas moduladas a una mayor frecuencia (1 GHz). De esta manera se comprueba que su comportamiento no varía en el rango de operación.

Las medidas obtenidas por el osciloscopio no se aprecian con exactitud (por la aparición de la parte transitoria). Sin embargo, se puede observar que ambas señales están solapadas y mantienen la misma forma.

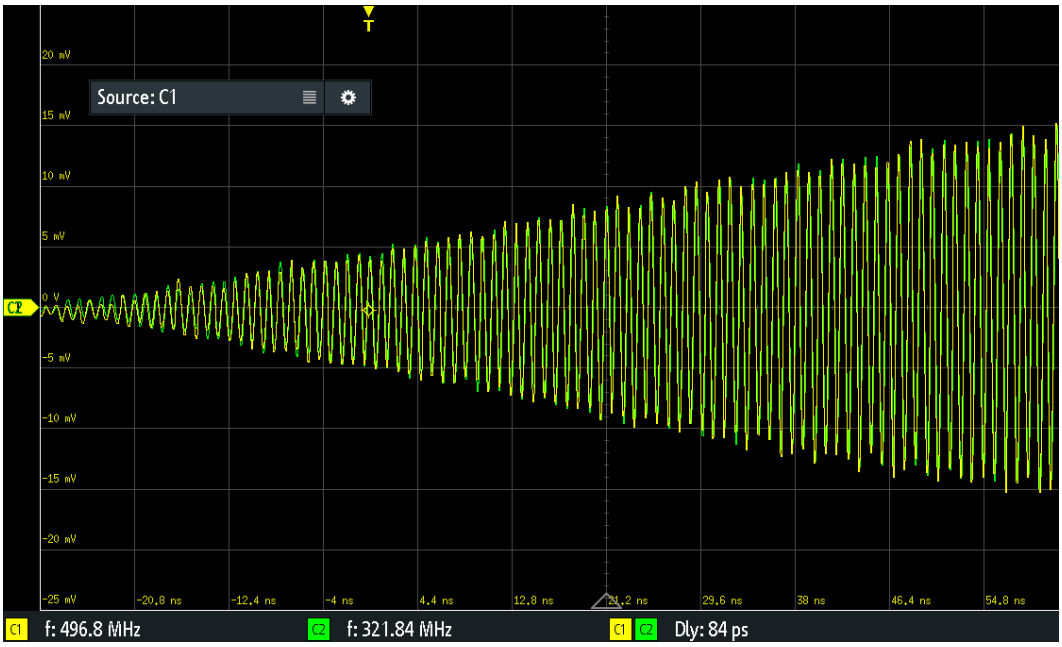

Figura 7-39: Desfase Señal pulsada DAC0-DAC1 (SYSREF) – 1 GHz

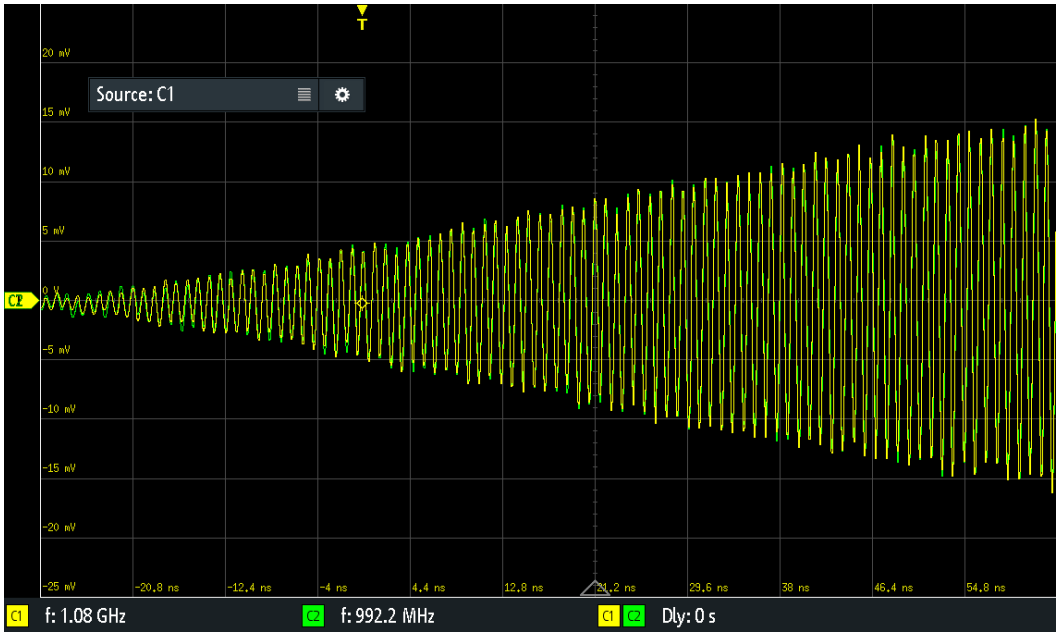

Figura 7-40: Desfase Señal pulsada DAC0-DAC4 (SYSREF) – 1 GHz

Al comprobar que en este escenario las señales no se encuentran desfasadas, se verifica que los resultados obtenidos a lo largo de esta fase de pruebas son válidos.

7.1.2 Pruebas con el analizador de espectros

Para comprobar con mayor claridad los resultados anteriores, se puede observar en el analizador de espectros cómo aparece el tono a la frecuencia deseada, además de su frecuencia imagen.

En la Figura 5-41, se observa la señal modulada a 1.15 GHz del DAC0. Esta señal se puede apreciar como un tono situado en el marcador seleccionado. Respecto a los demás tonos (considerados espurios) son la señal continua acoplada por el mismo analizador de espectros y la réplica incorporada a 2.04488 GHz.

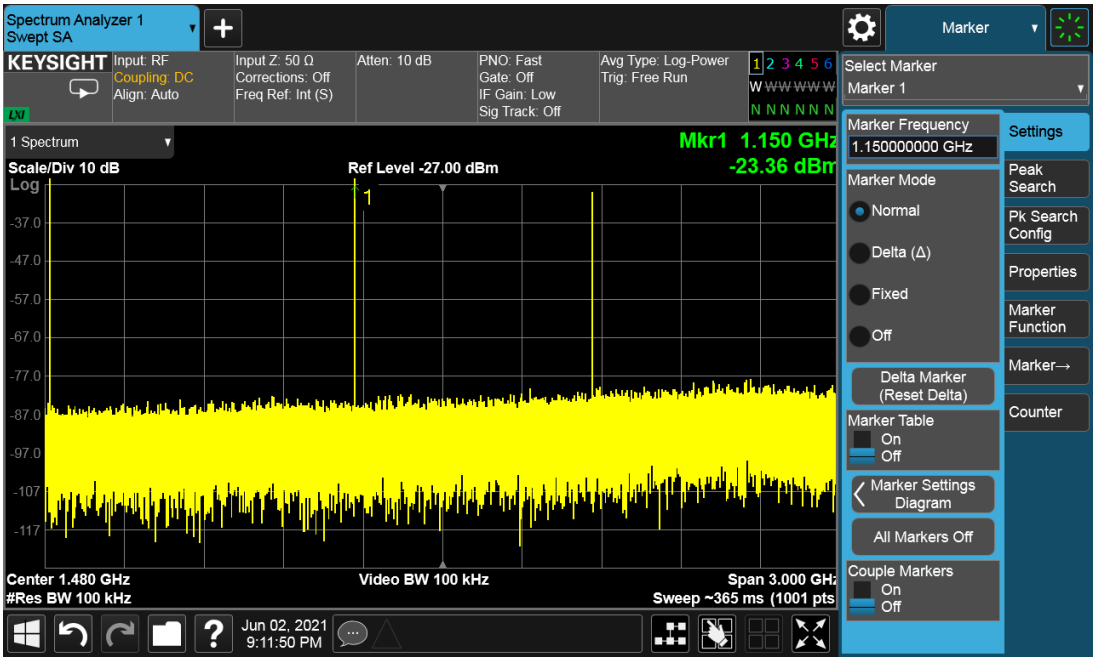

Figura 7-41: Analizador de espectros DAC0 – 1.15 GHz

En cambio, la Figura 7-42 representa cómo la señal modulada a 1.8 GHz supera la réplica, verificando que, aunque el osciloscopio no es capaz de representarlo (véase en la Figura 7-33 y la Figura 7-34), dicha señal sigue presente.

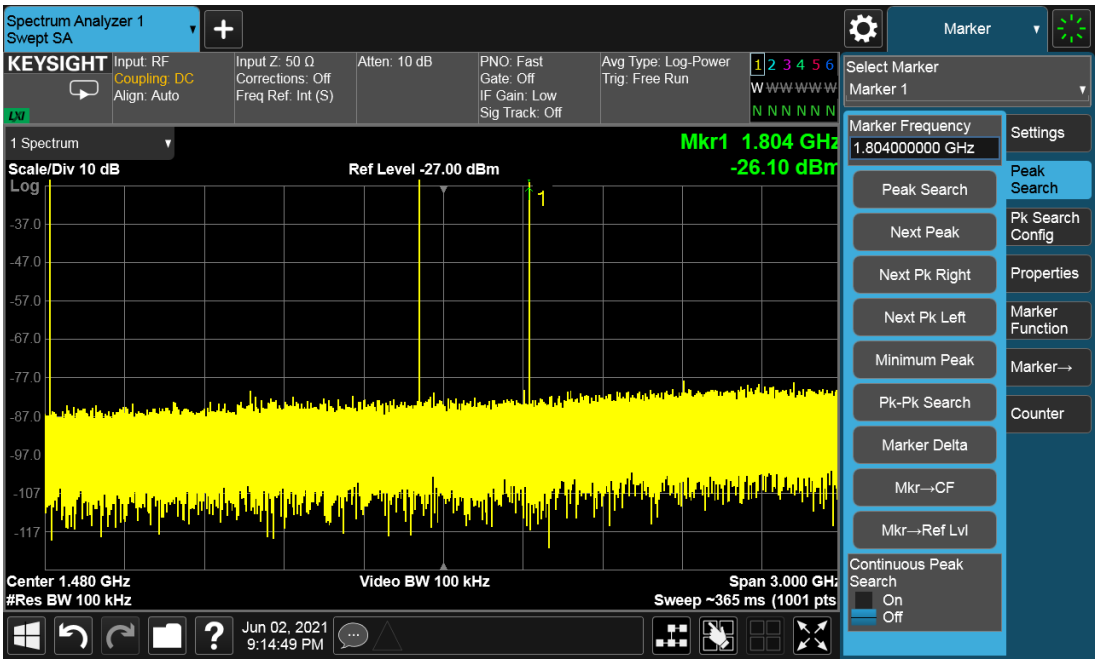

Figura 7-42: Analizador de espectros DAC0 – 1.8 GHz

La Figura 7-43, representa el mismo caso anterior, pero a través de un barrido de 10 GHz. En esta imagen se pueden apreciar las diferentes réplicas en todos los múltiplos de la frecuencia de muestreo.

Nótese que, aunque en la teoría matemática las réplicas deberían ser infinitas e idénticas en amplitud con la señal original, éstas se van atenuando por el efecto paso bajo del DAC. Conforme la señal aumenta en frecuencia, empiezan a cobrar importancia elementos parásitos y de segundo orden que van atenuando progresivamente dichas réplicas.

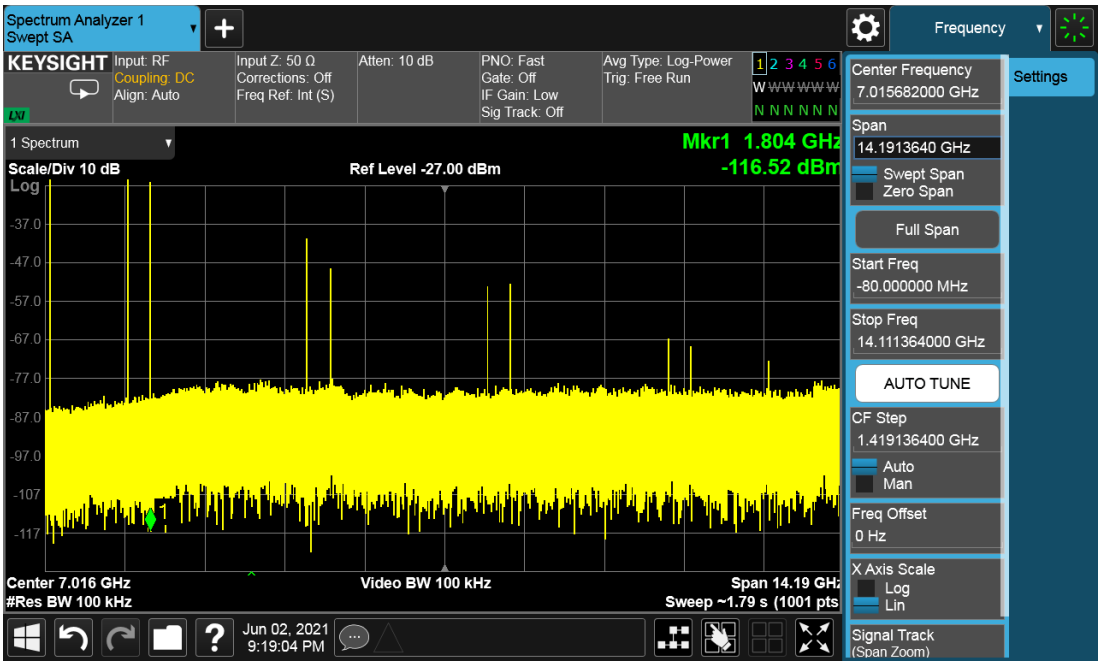

Figura 7-43: Analizador de espectros DAC0 – 1.8 GHz (barrido)

7.2 Resultados de la etapa de recepción

Debido a los inconvenientes que trae consigo trabajar con la placa "XM500 RFMC Balun" donde no vienen caracterizado los diferentes puertos, se obtienen unos resultados que no cumplen los requerimientos. Estos resultados se han conseguido a través de un analizador lógico (ILA).

Debido a que los resultados obtenidos, una vez aplicada la funcionalidad de "coherencia entre canales", no presentan condiciones válidas, se ha descartado la representación de los resultados logrados previamente de aplicar dicha funcionalidad.

7.2.1.1 Escenario: Señal sinusoidal

- **Pruebas con *MTS* (*multisincronización entre tiles*)**

ADCs de mismo *tile*

En esta primera prueba, se ha introducido una señal de 1 GHz a través del generador de señales. El procedimiento para realizarlo ha sido duplicar la señal a partir de un *splitter* de tal manera que alimente a dos ADCs. Mediante el mezclador se realiza la demodulación a 999 MHz, obteniendo una frecuencia en banda base de 1 MHz.

En la Figura 7-44, se puede observar en la parte superior, las dos señales pertenecientes al ADC0 (parte real y parte imaginaria), y en la parte inferior, la correspondiente al ADC1.

El resultado obtenido es que, aunque se está trabajando en un mismo *tile* (compartiendo el mismo reloj), existe un desfase de 8 muestras, que pasado a aspectos de tiempo corresponde a 20.03 ns (conociendo que la frecuencia del AXIS en Rx trabaja a 399.36 MHz para una muestra).

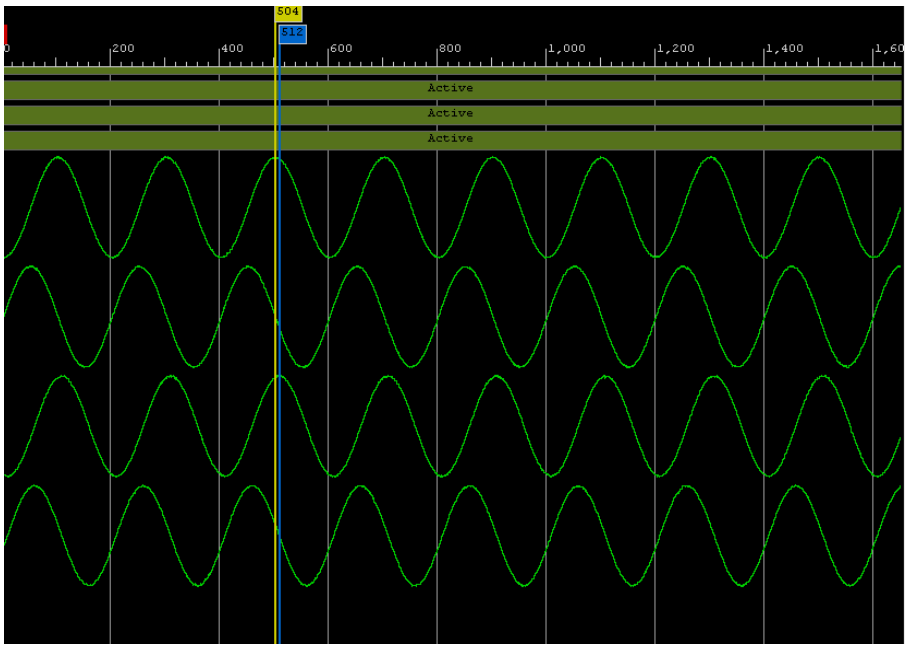

Figura 7-44: Desfase ADC0-ADC1 (SYSREF) – 1 MHz

ADCs de distinto *tile*

En esta segunda prueba se muestra el caso anterior, pero para ADCs de distinto *tile (*ADC0 y ADC2*)*, siendo las dos primeras señales (parte real y parte imaginaria) las equivalentes al ADC0 y las dos siguientes al ADC2.

El resultado en este caso (Figura 7-45) es un desfase de 29 muestras, es decir, 72.61 ns.

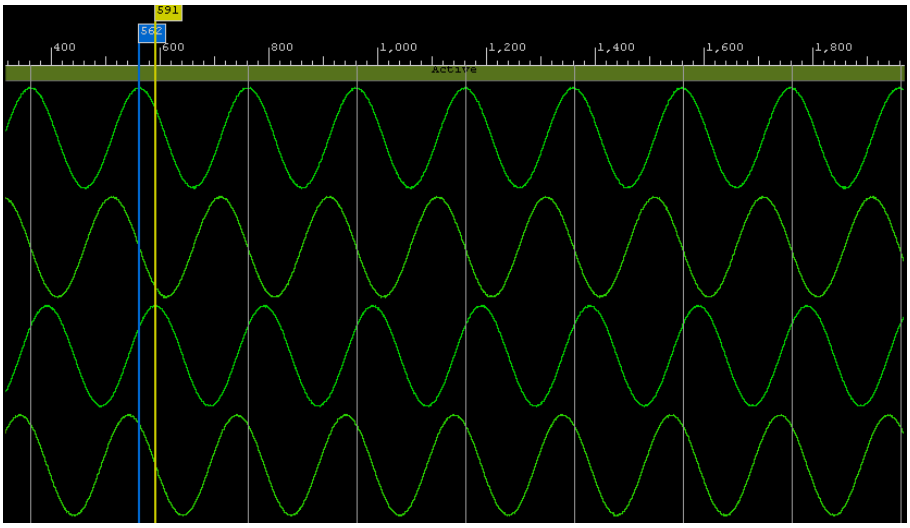

Figura 7-45: Desfase ADC0-ADC2 (SYSREF) – 1 MHz

Debido a que no se han conseguido unos resultados válidos a las especificaciones impuestas, se ha realizado un proceso de calibrado a nivel de software.

Proceso de calibrado

Para poder realizar este proceso es necesario que existan ciertos requerimientos, los cuales son:

- El desfase obtenido tenga una tendencia lineal, posibilitando la integración de una función que se adecue a cada frecuencia operada.
- Tener una conducta reproducible, es decir, el comportamiento obtenido de una sesión a otra sea muy similar.

El procedimiento se basa en introducir en la configuración del Mezclador del RFDC un valor "*PhaseOffset*" que elimine el desfase que impide cumplir el objetivo.

En la Figura 7-46, se muestra uno de los resultados del análisis tras la realización de diferentes pruebas para conocer que su comportamiento se repite en el tiempo (incluyendo el reseteo de la placa). En esta figura, se muestra el desfase de media obtenido entre dos canales del mismo *tile* (ADC0 y ADC1) frente a un barrido de frecuencias, siendo este de 500 a 1500 MHz.

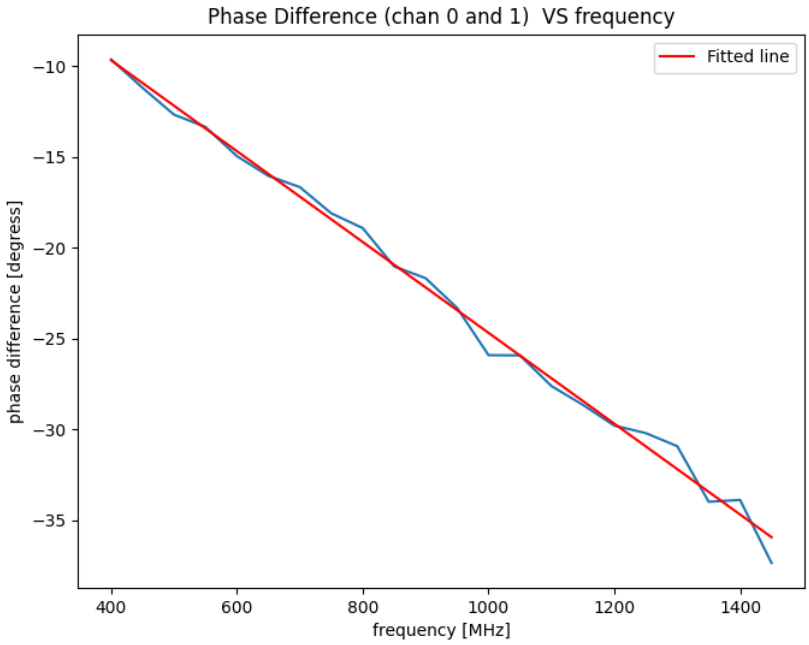

Figura 7-46: Resultados del análisis de desfase ADC0-ADC1

Conocido ese desfase proporcional a la frecuencia manejada, se realiza una función que dependiendo de la señal introducida en los ADCs determine el valor del parámetro *PhaseOffset*, obteniendo la compensación.

En la Figura 7-47, se muestran los resultados una vez introducida dicha calibración. El desfase obtenido es de 4 muestras, es decir, 10 ns.

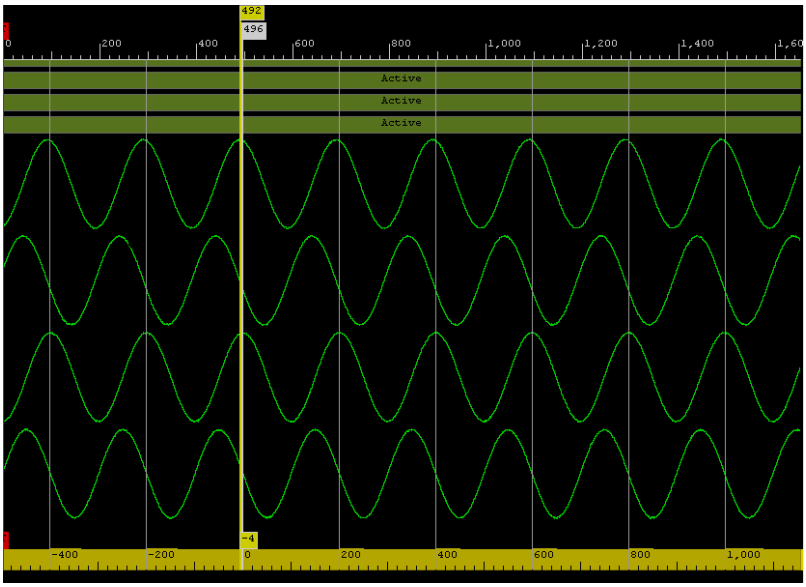

Figura 7-47: Desfase compensado ADC0-ADC1 – 1 MHz

Debido a que los resultados obtenidos no adquieren ese nivel de precisión que se quiere alcanzar, se descarta la opción para los demás casos. Esto intensifica la necesidad de una placa adaptada para cumplir el objetivo en esta etapa de recepción.

8 Conclusiones y trabajo futuro

En este apartado se detallan las conclusiones obtenidas a lo largo de este proyecto y se indican las posibles líneas futuras de investigación.

8.1 Conclusiones

En relación a la creación del diseño, se enumeran las siguientes conclusiones:

- Se ha elaborado el análisis de los diferentes requerimientos que se piden en el diseño. Para ello, se han realizado diferentes cálculos y comprobaciones para considerar que las características escogidas para su implementación son idóneas. Entre estos parámetros se encuentra la frecuencia de muestreo a 9.984 MHz para la etapa de transmisión, una tasa de muestreo que alcance el ancho de banda efectivo propuesto (en este caso 3.194 GSPS) para ambas etapas (Tx y Rx) y que, además, dicha tasa escogida esté acondicionada para que la generación del reloj sea precisa por parte de los sintetizadores (LMK04208 y LMX2594). Ambos sintetizadores se han configurado a partir del programa TICS Pro de la empresa *Texas Instruments*.
- Se ha escogido como placa de evaluación la ZCU111 RFSoC para la implementación de este proyecto, ya que proporciona una cadena de radiofrecuencia con 8 DACs y 8 ADCs con un alcance de tasa de muestreo mayor a la requerida, siendo su máxima tasa 6.554 GSPS y 4.096 GSPS respectivamente.
- Se ha utilizado de forma intensiva la herramienta Vivado para implementar el hardware a partir de los *IP Cores* proporcionados por su catálogo. Se ha necesitado invertir mucho tiempo de lectura en la documentación/guías que proporciona Xilinx para los diferentes módulos, necesario para comprender su funcionamiento y adecuarlo al uso que se quiere pretender.
- Se ha estudiado qué metodología es la más apropiada para alcanzar la tasa de muestreo a partir de las etapas de procesamiento de señal (interpolación y diezmado). Para este diseño, se ha elegido filtros FIR como la opción más adecuada (integrados en el módulo FIR Compiler del catálogo de Xilinx). También, para conocer la eficacia de los filtros escogidos (tras una fase de simulación), se han realizado pruebas de carácter menor implementándolos en la placa MPSoC.
- En la realización del diseño, se ha aplicado el formato F*low Control* de tipo "Blocking", es decir, la utilización en el bus de datos AXIS de señales *tvalid* y *tready* en todo momento, descartando la existencia de pérdidas "involuntarias" durante todo el proceso. Cabe destacar que se ha establecido una mejora en su optimización atendiendo tanto a aspectos de *throughput* como en el ahorro de recursos, no alcanzando el máximo soportado de la placa a esperas de incluir en un futuro nuevas operaciones que requieran de más requerimientos.
- Por último, se ha procedido a incluir una etapa de software mediante el *framework* PYNQ. Esta plataforma facilita mucho la integración de aplicaciones, además de facilitar al diseñador poder manejar el diseño configurando los módulos para realizar la fase de pruebas sin necesidad de volver a implementar un nuevo diseño. También, permite elegir qué banco de datos se quiere distribuir en el proceso y alcanzar la "coherencia entre canales" gracias a la integración de la funcionalidad *runMTS()*.

Respecto a la fase de resultados, antes de continuar con sus conclusiones, es necesario conocer que las medidas obtenidas han sido tomadas a partir del uso de conectores SMA y adaptadores BNC con las mismas características (longitud y fabricante). No tener esto en

cuenta, implica introducir retardos y desfases que imposibilitan alcanzar el nivel de precisión que se pretende. Una vez dicho esto, se enuncia lo siguiente:

- Se ha conseguido cumplir completamente 4 de las 5 especificaciones que se establecen en el diseño. Respecto a la especificación no desempeñada (*multisincronización entre tiles*), se ha podido resolver en parte, introduciendo en la etapa de transmisión, es decir, en el apartado de sus DACs, una tarjeta de adaptación proporcionada por la empresa SENER. Dicha placa caracteriza sus 8 canales de Tx con un desfase mínimo entre los diferentes puertos.
- Para la etapa de transmisión se ha trabajado con diferentes escenarios, además de haber realizado pruebas con las distintas modalidades que se pueden obtener del mezclador a la hora de modular la señal. Siendo EVENT_SRC_SYSREF, la modalidad que establece un reloj común para todos los tiles, obteniendo la coherencia para todos los canales. Sobre los valores del desfase conseguidos una vez introducida la funcionalidad de sincronización, han sido menores a 40 picosegundos, algo asequible a la hora de compensar tras la realización de un calibrado entre la ZCU111 y el producto destinado. Cabe destacar que, a falta de introducir un filtro paso bajo analógico que elimine las réplicas, se ha conseguido con éxito poder operar en el ancho de banda efectivo propuesto.
- Respecto a la etapa de recepción, por falta de una tecnología equivalente a la aplicada en la etapa de transmisión, no se ha podido alcanzar resultados concluyentes. Para intentar remediarlo, se ha realizado una compensación sin éxito a nivel de software. Esto es debido a que el nivel del calibrado que se puede obtener por software no alcanza el nivel de precisión que se quiere alcanzar. Los valores del desfase en este caso se mueven en torno a 10 nanosegundos (mucho mayores a los estipulados, 50 picosegundos).

Para terminar, se ha de comentar que la ZCU111, por defecto, no está construida con la finalidad de integrar aplicaciones específicas como la que se realiza en este proyecto. Esto se comprueba ya que trae consigo una tarjeta de conversión (XM500 RFMC Balun) para expandir los transceptores GTY en puertos SMA. Sin embargo, esta placa de conversión tiene dos formatos en la distribución de estos puertos (tanto para DACs como ADCs), donde solo la mitad de estos dispositivos se han instalado como conectores diferenciales convertidos a modo común (*singled-ended*) a través de la incorporación de un *balun*. Cabe destacar que el rango de frecuencias en el que opera también es distinto.

Intentar trabajar con estas características donde la longitud de pista es diferente, complica de gran manera la integración de este proyecto. Aunque, a falta de obtener una placa adaptada para esta etapa de recepción, el hecho de operar con las mismas características, compartiendo la misma frecuencia de evento común (SYSREF) y misma tasa de muestreo en ambas etapas, se replantea la posibilidad de que dicha coherencia pueda existir, cumpliendo con ello todos los objetivos.

Para concluir, en la Figura 8-1 y Figura 8-2 se muestra el uso de recursos utilizado para la implementación de este diseño.

Utilization Post-Synthesis | **Post-Implementation**

Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	98820	425280	23.24
LUTRAM	19469	213600	9.11
FF	210130	850560	24.70
BRAM	505	1080	46.76
DSP	2128	4272	49.81
IO	4	347	1.15
BUFG	7	696	1.01
MMCM	2	8	25.00

Figura 8-1: Utilización de recursos en la implementación

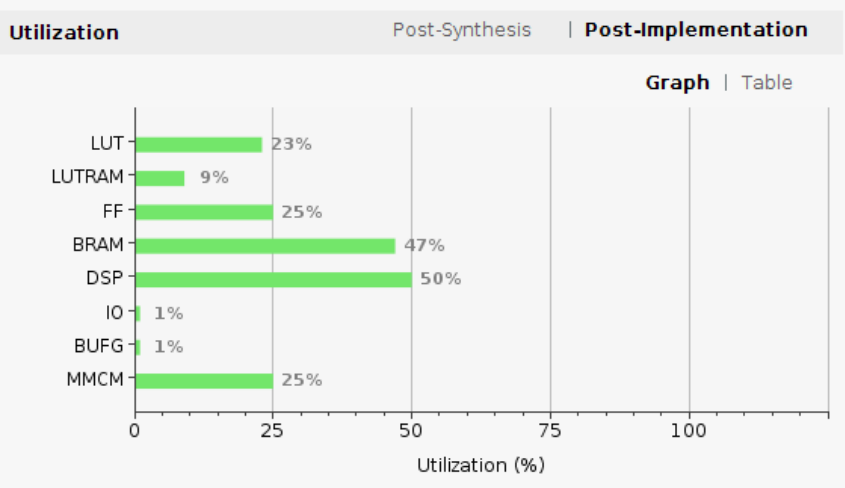

Figura 8-2: Utilización de recursos en la implementación (%)

8.2 Trabajo futuro

Este diseño, una vez comprobado y garantizado que todos los requerimientos son válidos, podría implementarse en placas de evaluación de generaciones más avanzadas. Como ya se contaba en la introducción de este documento, con la evolución de estas placas se han alcanzado niveles más sofisticados en su estructura, como es el aumento de canales o un mayor alcance en la tasa de muestreo, aportando una optimización en el procesamiento de señal (disminuyendo la carga computacional).

Con este diseño y las cualidades que definen la placa RFSoC, se pueden integrar aplicaciones para sistemas de comunicación de 5G, LTE, radar, etc. Gracias a la flexibilidad de estas placas, este diseño puede ser ajustado de manera que se adecúe a las características de cierto producto y, por tanto, poder interactuar con él realizando la fase de verificación previa para realizar su calibrado sin la necesidad de llevarlo al exterior, lo cual resulta de gran utilidad.

Poder operar con el apartado software a través de lenguajes de programación, permite construir escenarios aportando características concretas para poder realizar pruebas más exhaustivas. Por ejemplo, haciendo referencia a una aplicación basada en señales radar,

donde permite entrenar al receptor sin la necesidad de un radar primario y lo mismo para el proceso inverso.

Otra posibilidad es operar con dos RFSoC, una en la etapa de transmisión de un producto y otra en la etapa de recepción de otro para tratar la señal y adecuarla a las necesidades de la aplicación.

En conclusión, este diseño tiene multitud de posibilidades, tan solo es necesario realizar los debidos ajustes del mismo para cumplir un objetivo.

Referencias

Fuentes de recursos:

[1] https://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/ug1037-vivado-axi-reference-guide.pdf, Pág. 7

[2] https://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/ug1037-vivado-axi-reference-guide.pdf, Pág. 8

[3] https://documentation-service.arm.com/static/602a9df190ee6824a1e02b98?token=, Pág. 120

[4] https://documentation-service.arm.com/static/5f914c33f86e16515cdc2b25?token=

[5] https://documentation-service.arm.com/static/604f31721da8f8344a2c9f0f?token=

[6] https://documentation-service.arm.com/static/604f31721da8f8344a2c9f2a?token=

[7] https://documentation-service.arm.com/static/604f31721da8f8344a2c9f22?token=

[8] https://www.element14.com/community/themes/images/2019/avnet_rfsoc_kit_exploded.png

[9] https://media.digikey.com/Photos/Xilinx%20Photos/EK-U1-ZCU102-G.jpg

[10] ZCU102 Evaluation Board User Guide (xilinx.com), Pág. 23

[11] https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcTrRZ5S3v-jkT-mpwSI-Cw9E-Ns1gZaq0xpxA&usqp=CAU

[12] https://www.xilinx.com/support/documentation/ip_documentation/usp_rf_data_converter/v2_1/pg269-rf-data-converter.pdf, Pág. 49

[13] http://www.lucamartino.altervista.org/DIEZMADO_Interpolacion_Secuencia.pdf, Pág. 13

[14] http://www.lucamartino.altervista.org/DIEZMADO_Interpolacion_Secuencia.pdf, Pág. 9

[15] https://www.xilinx.com/support/documentation/boards_and_kits/zcu111/ug1271-zcu111-eval-bd.pdf, Pág. 54

[16] https://www.xilinx.com/support/documentation/ip_documentation/usp_rf_data_converter/v2_4/pg269-rf-data-converter.pdf, Pág. 174

[17] https://www.xilinx.com/support/documentation/ip_documentation/usp_rf_data_converter/v2_4/pg269-rf-data-converter.pdf, Pág. 176

[18] https://www.hackster.io/whitney-knitter/xilinx-dds-compiler-ip-tutorial-on-the-ultra96-f820db

[19] https://www.xilinx.com/support/documentation/ip_documentation/usp_rf_data_converter/v2_2/pg269-rf-data-converter.pdf, Pág. 17

[20] https://www.xilinx.com/support/documentation/ip_documentation/usp_rf_data_converter/v2_1/pg269-rf-data-converter.pdf, Pág. 8

[21] https://www.xilinx.com/support/documentation/ip_documentation/usp_rf_data_converter/v2_1/pg269-rf-data-converter.pdf, Pág. 8

[22] https://www.xilinx.com/support/documentation/ip_documentation/usp_rf_data_converter/v2_1/pg269-rf-data-converter.pdf, Pág. 75

[23] https://www.xilinx.com/support/documentation/ip_documentation/usp_rf_data_converter/v2_2/pg269-rf-data-converter.pdf, Pág. 65

[24] https://www.xilinx.com/support/documentation/ip_documentation/usp_rf_data_con

verter/v2_1/pg269-rf-data-converter.pdf, Pág. 36
[25] https://www.xilinx.com/member/forms/download/design-license.html?cid=9da5f26d-5d84-4a20-89d8-dc7437705c65&filename=zcu111-schematic-xtp508.zip, Pág. 4
[26] https://www.xilinx.com/support/documentation/ip_documentation/usp_rf_data_converter/v2_2/pg269-rf-data-converter.pdf, Pág. 126
[27] https://i1.wp.com/cdn-images-1.medium.com/max/1024/1*Nk6O4mGHDioZoF0aa7_VOg.png?w=1014&ssl=1

Enlaces en la documentación:

[28] https://www.xilinx.com/products/boards-and-kits/zcu111.html#documentation
[29] https://www.xilinx.com/support/documentation/ip_documentation/usp_rf_data_converter/v2_4/pg269-rf-data-converter.pdf, Pág. 178
[30] https://www.xilinx.com/support/documentation/ip_documentation/usp_rf_data_converter/v2_4/pg269-rf-data-converter.pdf, Pág. 173

Referencias de información:

[31] Tratamiento Digital de Señales: Principios, Algoritmos y Aplicaciones". Proakis, J.G.; Manolakis, D.G. Prentice Hall. (1998)
[32] Digital Signal Processing: A Computer-Based Approach, 2e". Mitra S. K, McGraw-Hill. (2001)
[33] Symmetry Exploitation in Digital Interpolators/Decimators, IEEE Transactions on Signal Processing, Mou, Zhi-Jian. (1996)
[34] Ian Kuon, R. T. "FPGA Architecture". (2008).
[35] Certain Topics in Telegraph Transmission Theory, H. Nyquist. (1928)
[36] Communication in the Presence of Noise, Claude Shannon (1949)
[37] Proceeding of the IEEE, Vol. 90, No. 2, (2002) - Certain topics in telegraph transmission theory - Proceedings of the IEEE (oregonstate.edu)
[38] http://fab.cba.mit.edu/classes/S62.12/docs/Shannon_noise.pdf
[39] https://www.aeroespacial.sener/productos-comint
[40] https://www.aeroespacial.sener/productos/soluciones-integradas-inteligencia-comunicaciones
[41] AMBA: https://developer.arm.com/architectures/system-architectures/amba/amba-4
[42] https://documentation-service.arm.com/static/602a9df190ee6824a1e02b98?token=
[43] https://forums.xilinx.com/t5/Design-and-Debug-Techniques-Blog/AXI-Basics-1-Introduction-to-AXI/ba-p/1053914
[44] https://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf
[45] https://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/ug1037-vivado-axi-reference-guide.pdf
[46] AXI4-Lite: https://developer.arm.com/documentation/ihi0022/latest
[47] AXI4-Stream: https://developer.arm.com/documentation/ihi0051/latest
[48] https://www.xilinx.com/products/boards-and-kits/zcu111.html#documentation
[49] Tabla RFSoC: Zynq UltraScale+ RFSoC (xilinx.com)
[50] Tabla MPSoC: Zynq UltraScale+ MPSoC (xilinx.com)
[51] Upsampling: https://www.wikiwand.com/en/Upsampling
[52] Downsampling:

https://www.wikiwand.com/en/Downsampling_(signal_processing)
[53] http://ocw.uv.es/ingenieria-y-arquitectura/filtros-digitales/tema_7_modificacion_de_la_frecuencia_de_muestreo.pdf
[54] Zynq PS: https://www.xilinx.com/support/documentation/data_sheets/ds891-zynq-ultrascale-plus-overview.pdf
[55] Zynq UltraScale+ RF Data Converter: https://www.xilinx.com/support/documentation/ip_documentation/usp_rf_data_converter/v2_2/pg269-rf-data-converter.pdf
[56] AXI Direct Memory Access: https://www.xilinx.com/support/documentation/ip_documentation/axi_dma/v7_1/pg021_axi_dma.pdf
[57] AXI Interconnect: https://www.xilinx.com/support/documentation/ip_documentation/axi_interconnect/v2_1/pg059-axi-interconnect.pdf
[58] AXI SmartConnect: https://www.xilinx.com/support/documentation/ip_documentation/smartconnect/v1_0/pg247-smartconnect.pdf
[59] DDS Compiler: https://www.xilinx.com/support/documentation/ip_documentation/dds_compiler/v6_0/pg141-dds-compiler.pdf
[60] AXI4-Stream Switch/ Data Width Converter/ Combiner/ Register Slice: https://www.xilinx.com/support/documentation/ip_documentation/axis_infrastructure_ip_suite/v1_1/pg085-axi4stream-infrastructure.pdf
[61] AXI Interrupt Controller: https://www.xilinx.com/support/documentation/ip_documentation/axi_intc/v4_1/pg099-axi-intc.pdf
[62] FIR Compiler: https://www.xilinx.com/support/documentation/ip_documentation/fir_compiler/v7_2/pg149-fir-compiler.pdf
[63] https://es.mathworks.com/help/dsp/ug/design-of-decimatorsinterpolators.html
[64] https://es.mathworks.com/help/signal/ref/designfilt.html
[65] AXI4-Stream Data FIFO: https://www.xilinx.com/support/documentation/ip_documentation/axi_fifo_mm_s/v4_1/pg080-axi-fifo-mm-s.pdf
[66] Clocking Wizard: https://www.xilinx.com/support/documentation/ip_documentation/clk_wiz/v6_0/pg065-clk-wiz.pdf
[67] Utility Vector Logic: https://www.xilinx.com/support/documentation/ip_documentation/util_vector_logic/v2_0/ds913.pdf
[68] Processor System Reset Module: https://www.xilinx.com/support/documentation/ip_documentation/proc_sys_reset/v5_0/pg164-proc-sys-reset.pdf
[69] Utility Buffer: https://www.xilinx.com/support/documentation/ip_documentation/proc_sys_reset/v5_0/pg164-proc-sys-reset.pdf
[70] AXI GPIO: https://www.xilinx.com/support/documentation/ip_documentation/axi_gpio/v2_0/pg144-axi-gpio.pdf
[71] System ILA:

https://www.xilinx.com/support/documentation/ip_documentation/system_ila/v1_0/pg261-system-ila.pdf
[72] TICS Pro: https://www.ti.com/tool/TICSPRO-SW
[73] https://www.ti.com/lit/ug/snau200/snau200.pdf?ts=1624182153700
[74] Librerías PYNQ: https://github.com/Xilinx/ZCU111-PYNQ/tree/v2.5
[75] https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/57606309/ZCU111+RFSoC+RF+Data+Converter+Evaluation+Tool+Getting+Started+Guide
[76] https://www.digikey.es/es/articles/build-and-program-fpga-based-designs-quickly-python-jupyter-notebooks

Glosario

ADC	Conversor Analógico a Digital
AMBA	Advance Microcontroller Bus Architecture
API	Application Programming Interface
AXI	Advanced Extensible Interface
AXIL	AXI4-Lite
AXIS	AXI4-Stream
DAC	Conversor Digital a Analógico
DMA	AXI Direct Memory Access
EPS	Escuela Politécnica Superior
FF	Flip-Flop
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
MTS	Multisincronización entre tiles
PL	Lógica Programable
PS	Sistema de Procesamiento
RFDC	Zynq Ultrascale+ RF Data Converter
RX	Recepción
SoC	System on a Chip
STA	Static Timing Analysis
TX	Transmisión
UAM	Universidad Autónoma de Madrid

Anexos

A TICS Pro (Texas Instruments)

En este anexo se muestra las opciones introducidas en el programa TICS Pro para la obtención de registros que configuran los generadores de reloj externos al PL.

Estos generadores de reloj son: LMK04208 y LMX2594.

Antes de su configuración es necesario conocer que el LMX2594 es un sintetizador fraccional, el cual permite obtener una resolución de mHz.

Si se quieren obtener las mejores prestaciones a la hora de su generación (como es el objetivo en este diseño), es necesario que la frecuencia buscada sea múltiplo al reloj de la PFD (*phase frequency detector*). Siendo esta frecuencia (*Fpd*) de 122.88 MHz.

El hecho de no hacerlo puede generar ciertos espurios fraccionales o subfraccionales que estropearían la obtención de una señal precisa.

Por tanto, una posible solución sería modificar la PFD en relación a la tasa de muestreo escogida. Este proceso no es trivial, no se puede modificar de forma arbitraria, está relacionada con el LMK (el cual es un sintetizador entero), que a su vez trae consigo un reloj base de 122.88 MHz. Por tanto, para variar este reloj base es obligatorio hacerlo físicamente. Como en este diseño no se especifica una tasa en concreto, esta modificación no ha sido necesaria.

Conociendo esto anterior, la tasa de muestreo elegida para el diseño es 3.19488 GSPS (superior a los 3 GSPS requeridos en las especificaciones).

LMK04208

En la Figura 0-1, se muestra la configuración del "CLKins y PLLs" para la obtención del VCO externo. En este apartado no ha hecho falta modificar nada. Tan solo ser conscientes de que el valor del VCO es el mismo que el reloj base (122.88 MHz).

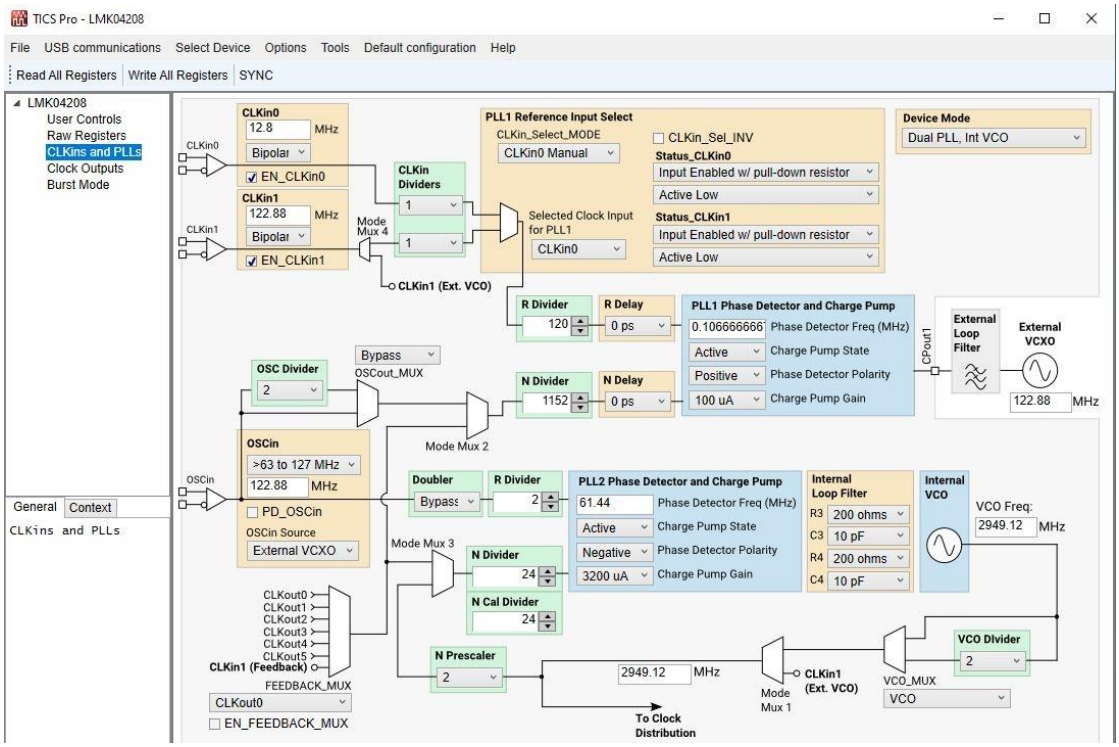

Figura 0-1: Configuración LMK04208 parte 1

La Figura 0-2 muestra la configuración para la obtención de la señal SYSREF (3.072 MHz) y de la señal PL_CLK. Viene en relación a la Figura 5-11.

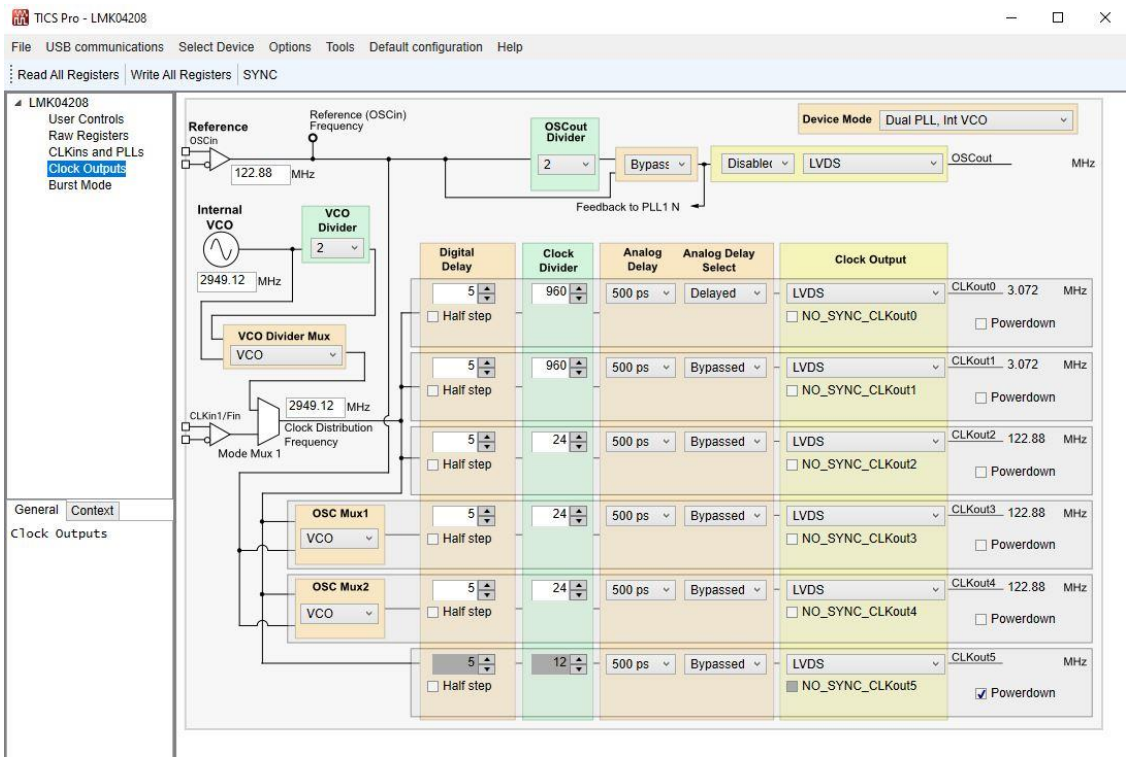

Figura 0-2: Configuración LMK04208 parte 2

Los registros obtenidos del LMK04208 son los siguientes:

```
122.888: [
0x00160040, 0x20147800, 0x00147801, 0x00140302,
0x00140303, 0x00140304, 0x80140185, 0x01100006,
0x01100007, 0x01010008, 0x55555549, 0x9102410A,
0x0401100B, 0x1B0C006C, 0x2302806D, 0x0200000E,
0x8000800F, 0xC1550410, 0x00000058, 0x02C9C419,
0x8FA8001A, 0x10001E1B, 0x0021201C, 0x0180031D,
0x0200031E, 0x003F001F
]
```

LMX2594

En la Figura 0-3, se muestra la configuración realizada para el LMX. Se debe tener en cuenta lo anteriormente comentado, la *Fpd* debe reflejarse como 122.88 MHz.
Para la obtención de la tasa de muestreo escogida, *RFoutA/RFoutB* debe tener asignado dicho valor.

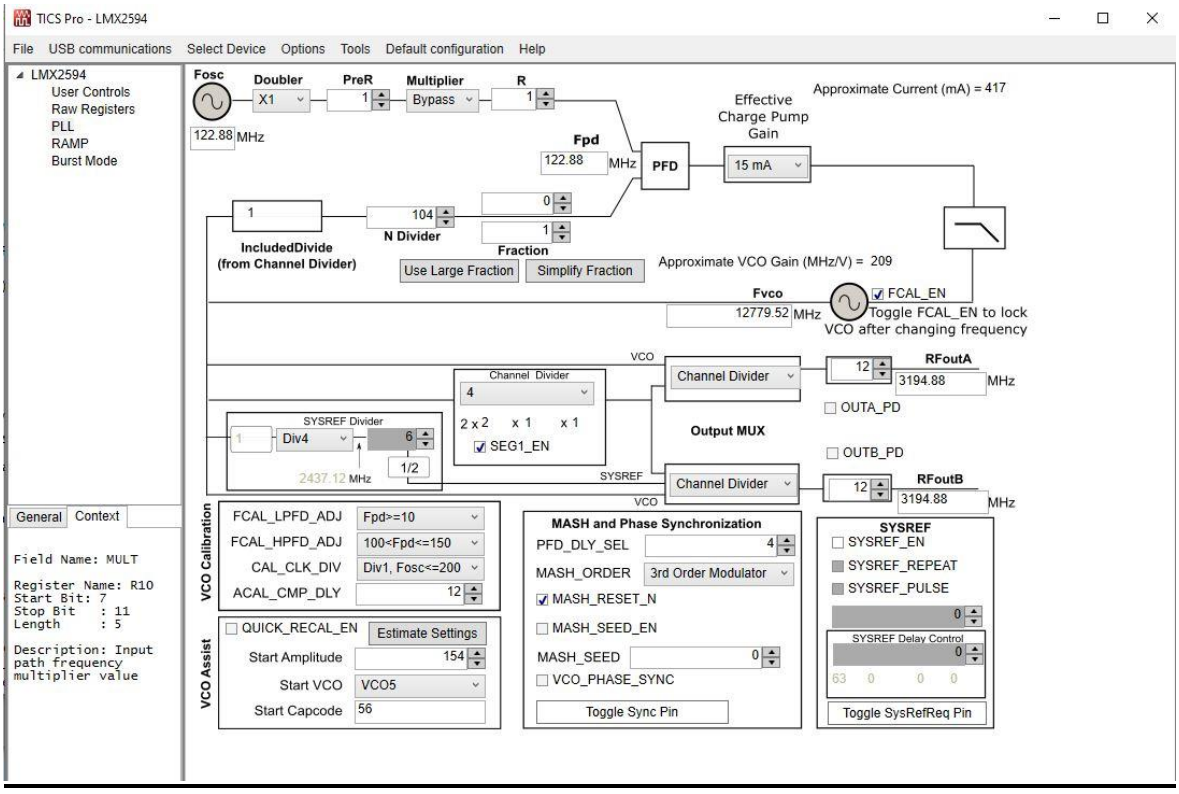

Figura 0-3: Configuración LMX2594

Los registros obtenidos del LMX2594 para la tasa de muestreo escogida son los siguientes:

```
3194.88: [
0x700000, 0x6F0000, 0x6E0000, 0x6D0000, 0x6C0000, 0x6B0000, 0x6A0000,0x690021,
0x680000, 0x670000, 0x663F80, 0x650011, 0x640000, 0x630000, 0x620200, 0x610888,
0x600000, 0x5F0000, 0x5E0000, 0x5D0000,0x5C0000, 0x5B0000, 0x5A0000, 0x590000,
0x580000, 0x570000, 0x560000, 0x55D300, 0x540001, 0x530000, 0x521E00, 0x510000,
0x506666, 0x4F0026, 0x4E0003, 0x4D0000,0x4C000C,0x4B0840,0x4A0000,0x49003F,
0x480001, 0x470081, 0x46C350, 0x450000, 0x4403E8, 0x430000, 0x4201F4, 0x410000,
0x401388, 0x3F0000, 0x3E0322, 0x3D00A8,0x3C0000,0x3B0001, 0x3A8001, 0x390020,
0x380000, 0x370000, 0x360000, 0x350000, 0x340820, 0x330080, 0x320000, 0x314180,
0x300300, 0x2F0300, 0x2E07FC, 0x2DC0CC,0x2C0C23,0x2B0000,0x2A0000,0x290000,
0x280000, 0x270001, 0x260000, 0x250404, 0x240068, 0x230004, 0x220000, 0x211E21,
0x200393, 0x1F43EC, 0x1E318C,0x1D318C,0x1C0488,0x1B0002,0x1A0DB0,0x190624,
0x18071A, 0x17007C, 0x160001, 0x150401, 0x14E048, 0x1327B7, 0x120064, 0x11012C,
0x100080, 0x0F064F, 0x0E1E70, 0x0D4000,0x0C5001,0x0B0018, 0x0A10D8, 0x090604,
0x082000, 0x0740B2, 0x06C802, 0x0500C8, 0x040C43, 0x030642, 0x020500, 0x010808,
0x00249C
]
```

B PYNQ

Es un proyecto de *open-source* de Xilinx, el cual permite la interacción del usuario con el diseño implementado. Se trata de una solución para aquellos desarrolladores que tienen poca experimentación en el mundo de las FPGAs. Da la posibilidad de implementar de forma rápida diseños, y así, conseguir un atajo significativo para aplicaciones que contienen un uso complejo en la configuración del hardware.
Las principales placas que soportan PYNQ son las tarjetas PYNQ-Z1/Z2, la Ultra96, la ZCU104 y la ZCU111.

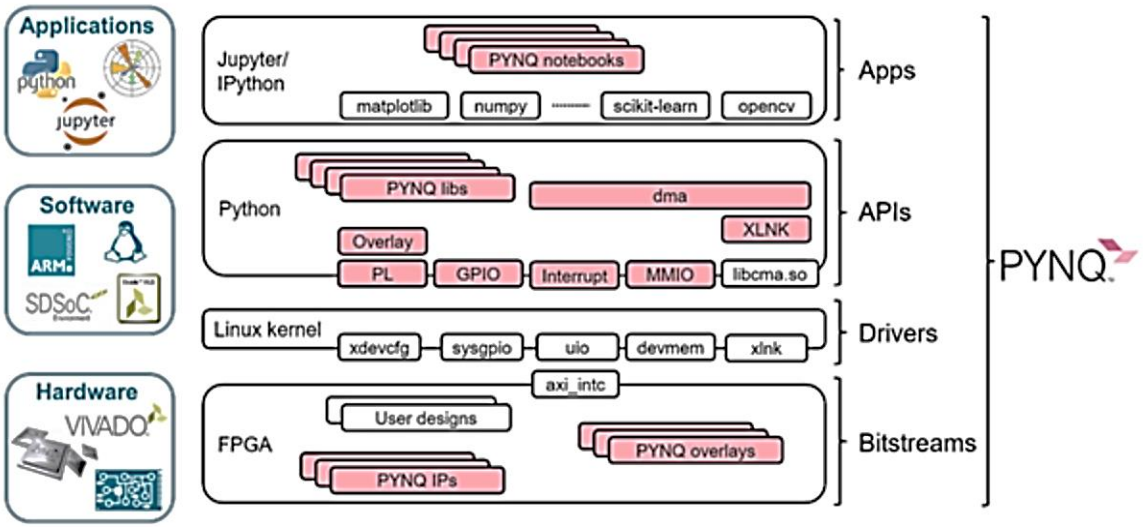

Figura 0-4: Funcionalidad de PYNQ

Esto permite simplificar tareas tanto para PS como PL. A nivel de usuario, desde el comienzo se ha buscado una manera de conseguir un rendimiento óptimo en el desarrollo de una FPGA convencional, la aparición de este *framework* elimina esta dificultad adyacente.

El entorno PYNQ se fundamenta por el uso de desarrollo del lenguaje de programación *Python*, siendo este, uno de los lenguajes punteros actuales. Se trata de un código abierto de libre acceso que tiene la gran ventaja de añadir diversidad y apoyo gracias a la exploración de repositorios de módulos creados por la comunidad. Donde en algunos casos, son librerías de software esenciales de funcionamiento para las tarjetas de evaluación (véase en este caso).

Procedimiento
La etapa de inicio (*boot*) conlleva, una vez implementado el diseño (a partir de la generación del *bitstream*), la división del flujo de bits. Estas divisiones se encapsulan en las distintas librerías de software implementadas, configurando la red del PL.
El paso siguiente repercute a manos del desarrollador, ya que será este, el responsable de integrar la funcionalidad buscada.

PYNQ trabaja sobre *Notebook Jupyter* (otro proyecto de código abierto). Este agrega la funcionalidad de trabajar los algoritmos alternando líneas de código para mostrar su representación, de manera que se puede realizar de forma efectiva su verificación.

Una vez formado el *Notebook Jupyter* a manos del usuario, este se traduce en un navegador web en formato HTTP y su combinación a partir de protocolos *WebSockets* para el contenido de memoria (tanto estático como dinámico).
Sobre el *back-end*, el servidor se comunica con el *core* de ejecución (*kernel*) utilizando un protocolo de mensajería (*ZeroMQ*).

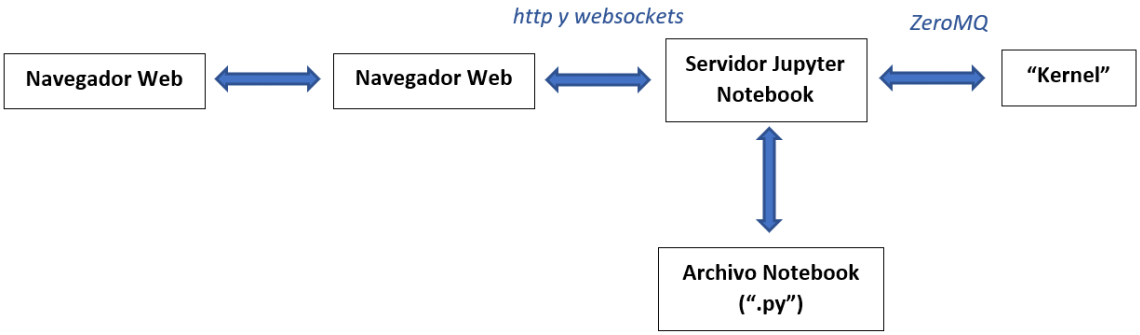

Figura 0-5: Procedimiento de PYNQ

Instalación de PYNQ
Para trabajar con PYNQ es necesario una serie de elementos:

- La placa en sí con la compatibilidad de PYNQ.
- Cable Ethernet para la comunicación entre diseñador-placa.
- Una tarjeta SD con memoria mayor a 8GB, donde se incluirá la imagen de arranque (en este proyecto se ha trabajado con la versión de PYNQ v2.5).

9 Apéndice

9.1 Código “Cálculo de la SYSREF”

```
#include <stdio.h>

//FORMATO: MHZ
int frec, freq1, freq2, freq3, freq4, freq5;

int mcd(freq1, freq2)
{
    if (freq2 == 0)
        return freq1;
    return mcd(freq2, freq1 % freq2);
}
int main()
{

    frec = 3194.88e3; // MHz -> SAMPLING RATE

    freq1 = frec / 16;
    freq2 = freq1;

    freq3 = 122.88e3; // MHz -> PL CLK
    freq4 = 199.68e3; // MHz -> AXIS ADC
    freq5 = 79.872e3; // MHz -> AXIS DAC

    freq2 = mcd((int)freq1, (int)freq2);

    int min = 1;
    int limit = 10001;

    while (2)
    {
        limit = limit - min;

        // PARAMETROS: GCD (SR), PL CLK, PL CLK'
        if (freq1 % limit == 0 && freq2 % limit == 0 && freq3 % limit == 0 &
& freq4 % limit == 0 && freq5 % limit == 0 && limit <= 10e3)
        {

            printf("Valor de la SYSREF es %lf MHz", limit / 1e3); // PARA QU
E SEAN MHz
            return 0;
        }
    }
}
```

9.2 Cálculo de los coeficientes FIR (Matlab)

```
L = 2;
M = 5;

format long

firInterp = dsp.FIRInterpolator(L,designMultirateFIR(L,1));
firDecim = dsp.FIRDecimator(M,designMultirateFIR(1,M));

% Interpolación o Diezmado
b=coeffs(firInterp)
coef=b.Numerator;

% Gráfica
hfv = fvtool(firInterp);
legend(hfv,"Filtro de Interpolación L="+num2str(L));

% Creación del fichero .coe (con formato Vivado)
sprintf("Numero de coeficientes para interpolacion grado x%d : %d\n", L,
length(coef))
fileID = fopen('coef_inter_2.coe','w');
fprintf(fileID,"RADIX=10;\n");
fprintf(fileID,"COEFDATA = ");
fprintf(fileID,"%8f,\n",coef);
fclose(fileID);
```

Respuesta en frecuencia de los filtros introducidos en el diseño:

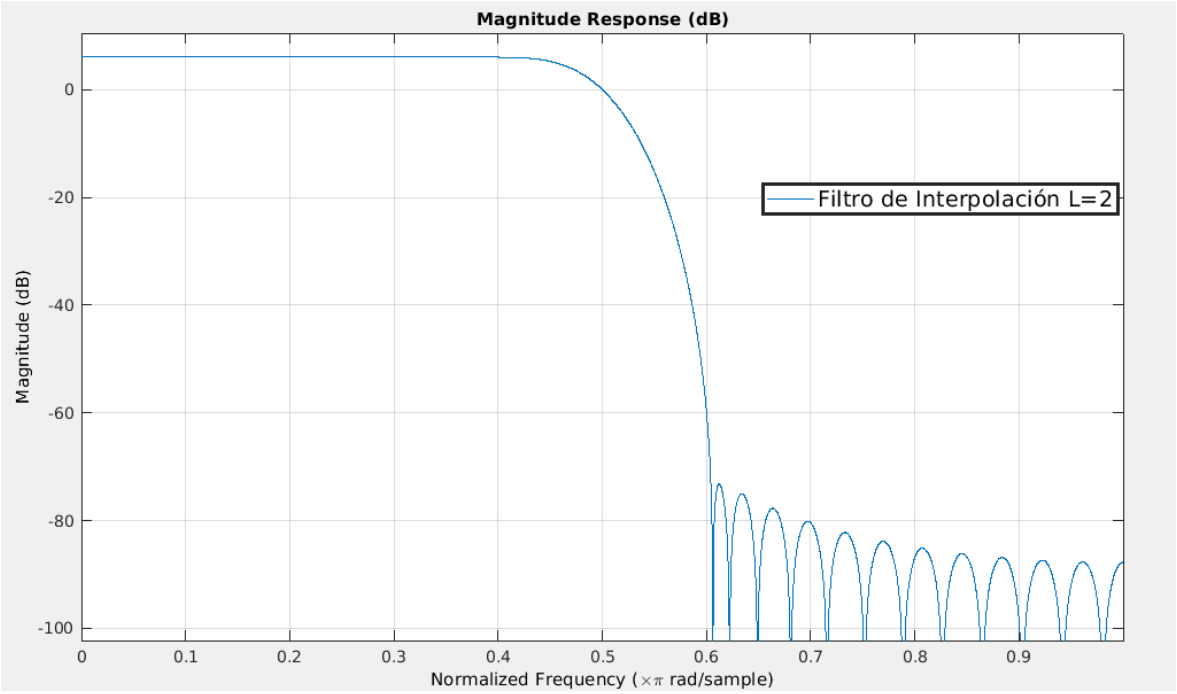

Figura 9-1: Respuesta en frecuencia Filtro interpolador (L=2)

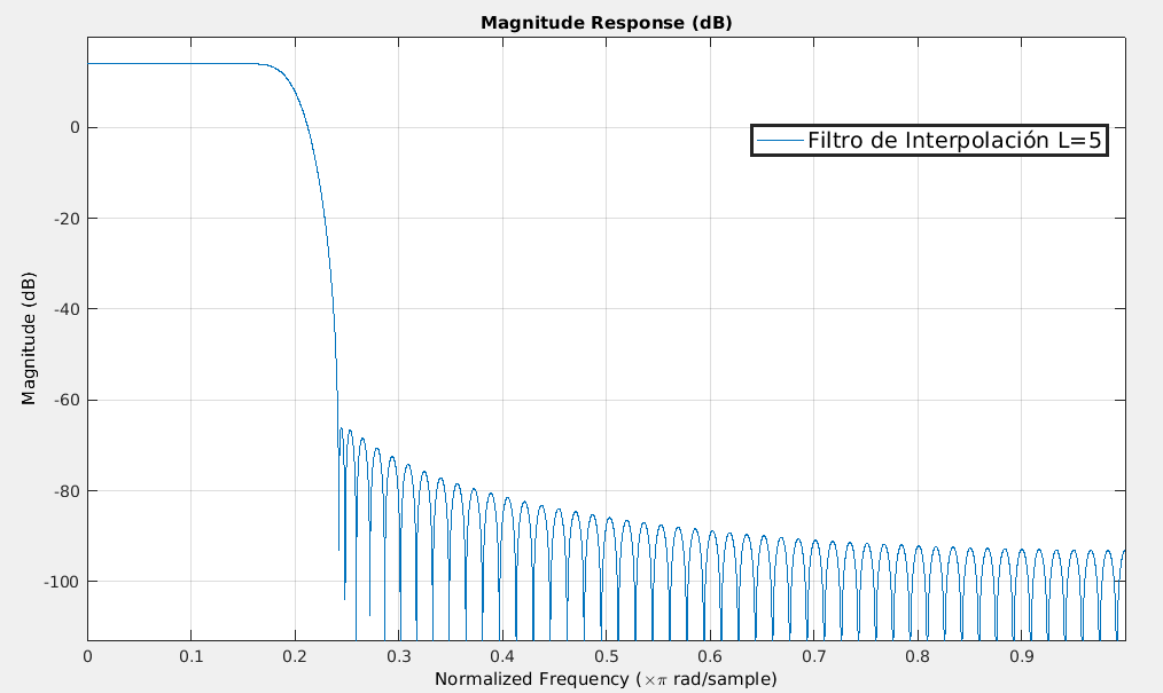

Figura 9-2: Respuesta en frecuencia Filtro interpolador (L=5)

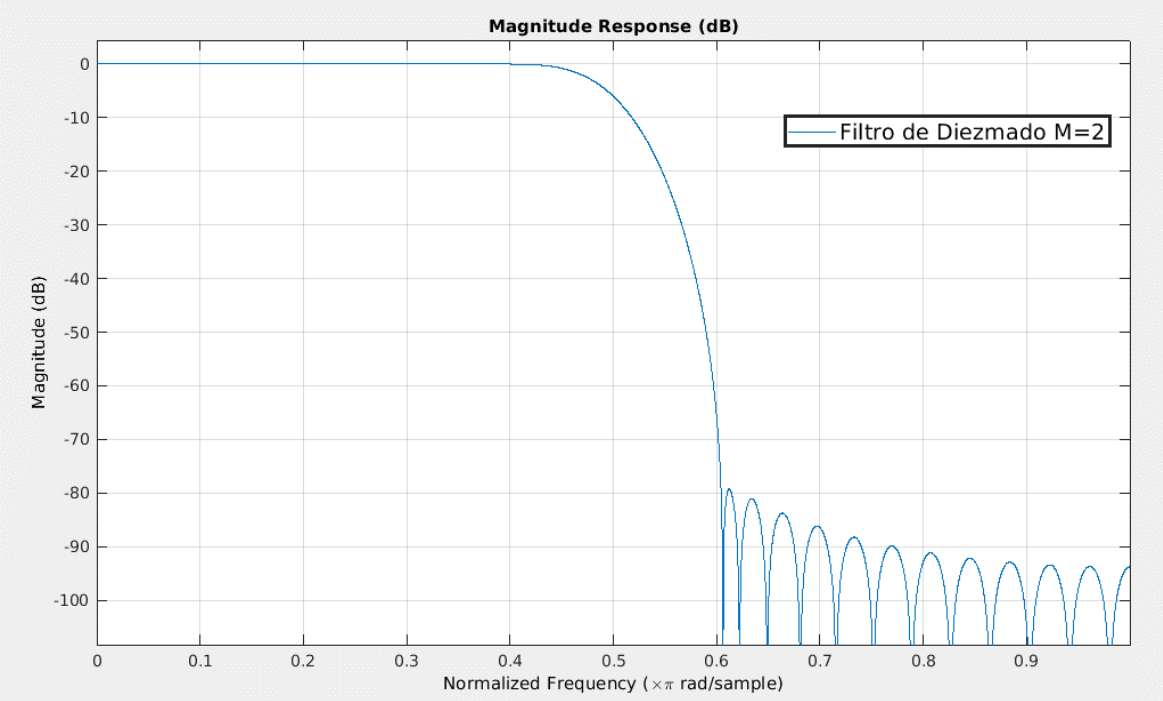

Figura 9-3: Respuesta en frecuencia Filtro diezmador (M=2)

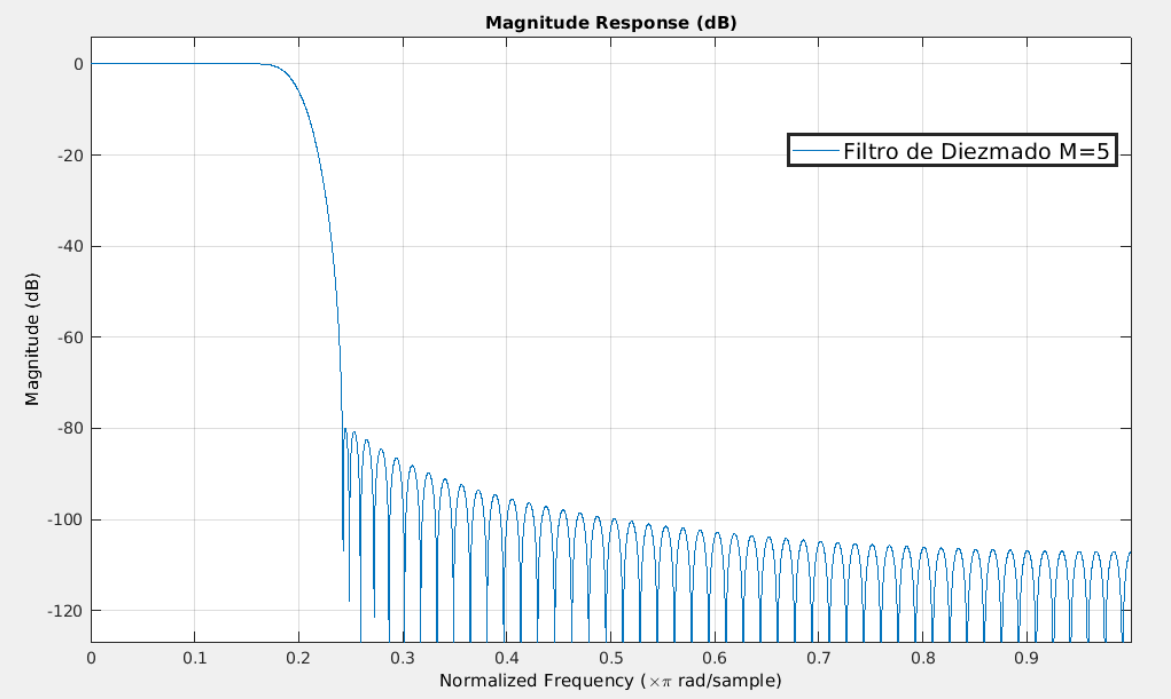

Figura 9-4: Respuesta en frecuencia Filtro diezmador (M=5)